

AD_____

Award Number: W81XWH-04-C-0067

TITLE: Medical Emergency Team Tutored Learning Environment

PRINCIPAL INVESTIGATOR: Eric A. Domeshek, Ph.D.

CONTRACTING ORGANIZATION: Stottler Henke Associates, Inc.
San Mateo, CA 94404

REPORT DATE: May 2008

TYPE OF REPORT: Final, Phase II

PREPARED FOR: U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

DISTRIBUTION STATEMENT: Approved for Public Release;
Distribution Unlimited

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision unless so designated by other documentation.

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 01-05-2008		2. REPORT TYPE Final, Phase II		3. DATES COVERED (From - To) 15 NOV 2004 - 14 APR 2008	
4. TITLE AND SUBTITLE Medical Emergency Team Tutored Learning Environment				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER W81XWH-04-C-0067	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) Eric A. Domeshek, Ph.D.				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Stottler Henke Associates, Inc. San Mateo, CA 94404				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Medical Research and Materiel Command Fort Detrick, Maryland 21702-5012				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT Stottler Henke investigated the design and implementation of a simulation-based Intelligent Tutoring System to teach appropriate decision-making and team-coordination skills focused on patient diagnosis, treatment, and management. Our Medical Emergency Team Tutored Learning Environment (METTLE) tutors individual medical personnel on the decisions and team interactions appropriate to their roles in front-line medicine. To enable any-time/any-where use, METTLE has been built as a rich web application that requires only standard web browser capabilities. To enable use by individual students, METTLE simulates other team members, including especially discussions with other actors. To address decision-making, METTLE exploits Socratic-style dialogs that elicit decision rationale. The main products of this effort include (1) software tools that support medical decision-making training scenarios, and applications in other decision-making domains emphasizing data-gathering and discussion, (2) a proof-of-concept training scenario that demonstrates the applicability of our technology to medical decision making in the context of Chemical, Biological, and Radiological emergencies, and (3) a methodology, tools, and set of content assets that can be used to speed construction of additional medical training scenarios. With increasing threats to U.S. military and civilians, deployable computer-based simulations with embedded intelligent tutors can lower costs, increase availability, and ensure uniformity of critical medical training.					
15. SUBJECT TERMS Simulation-Based Training, Medical Training, WMD/CBR Threat Response Intelligent Tutoring Systems, Dialogue, Artificial Intelligence					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			USAMRMC
			UU	133	19b. TELEPHONE NUMBER (include area code)

Table of Contents

1	Introduction.....	7
1.1	Purpose of the Research	7
1.2	Brief Description of the Research	7
1.3	Research Findings or Results	8
2	Body.....	9
2.1	Problem, Solution, Innovations, and Objectives	9
2.1.1	Problem Being Addressed.....	9
2.1.2	Overview of Solution	11
2.1.3	Innovations.....	12
2.1.4	Phase II Technical Objectives	13
2.2	Project Results	14
2.2.1	Tool Framework Modules and Concepts	14
2.2.1.1	SimVentive Extensions	14
2.2.1.2	GristEdit Extensions.....	14
2.2.1.3	Spreadsheet-Based Authoring	15
2.2.2	Final System Architecture.....	15
2.2.3	System Runtime	17
2.2.4	Sample Scenario.....	19
2.2.5	Authoring Capabilities	25
2.2.6	Authoring Methodology	25
2.2.6.1	Course Design: Roles, Curriculum, Scenarios, Concepts, and Reference Materials....	25
2.2.6.2	Conventional Behaviors, Forms, and Related Interactors	27
2.2.6.3	High-Level Scenario Design: Situation, Patient(s), Points, Agents, Scenes	28
2.2.6.4	Detailed Scenario Design: States, Evidence, Flows	29
2.2.6.5	Basic Scenario Behaviors.....	30
2.2.6.6	Scenario Media: Briefings, Images, Test Results, Sounds	31
2.2.6.7	Tutor Behaviors	32
2.2.6.8	Extended Agent-Initiative Discussions	33
2.2.6.9	Control Behaviors.....	34
2.2.6.10	Testing and Refinement.....	35
2.2.6.11	Summary of Authoring Costs	35
2.2.7	Final System Evaluation.....	35
2.2.7.1	Evaluation Method	35
2.2.7.2	Evaluation Results	36
2.2.8	Cost/Benefit Analysis.....	39
3	Key Research Accomplishments	42
4	Reportable Outcomes.....	43
5	Conclusions.....	44
6	References.....	46
7	Appendix A: Project Activities.....	48

8	Appendix B: Final System Architecture	55
8.1	The Enact Simulation Behavior Layer [Layer 7]	55
8.2	The Discuss Layer [Layer 6]	58
8.3	The GRAIN Layer [Layer 5]	62
8.4	The Interpreter Layer [Layer 4]	64
8.5	The Concept Layer [Layer 3]	65
9	Appendix C: Sample METTLE Scenario Interaction	66
10	Appendix D: Authoring Tools	86
10.1	Content Types and Organization (Directory Structure).....	86
10.2	Web-Based Authoring Tools	89
10.3	Spreadsheet-Based Behavior Authoring.....	93
10.4	Text-Editing for Authoring.....	96
11	Appendix E: Sample Agent Script Lines	98
11.1	Simple “What’s your name” Script Line.....	98
11.2	Asking About the Chief Complaint (With Tutor Commentary).....	99
11.3	A Curriculum-Relevant Behavior with Proactive Tutor Prompting.....	100
11.4	A Generalized Cue with Negative Tutor Feedback.....	100
11.5	A Complex Test with Positive Tutor Feedback.....	101
11.6	A Pure Control Script Line.....	102
12	Appendix F: Sample Tutor Dialog.....	103
12.1	A First Simple Unconditional Topic	103
12.2	A Topic with Repeated Probes and Reference to Concept Spaces.....	105
12.3	A Topic with Nested Point Structure.....	106
13	Appendix G: METTLE Help Pages Explaining System Use.....	109
13.1	METTLE On-Line Help Index	109
13.2	Launching METTLE	111
13.3	Logging-In to METTLE	112
13.4	Choosing a METTLE Scenario	115
13.5	Getting a METTLE Briefing	116
13.6	Playing a METTLE Scenario	117
13.7	Conversing with a Character	119
13.8	Accessing Patient Charts	120
13.9	Examining a Patient.....	121
13.10	Ordering Patient Tests	122
13.11	Ordering Patient Treatments.....	123
13.12	Patient Management Decisions	124
13.13	Emergency Systemic Response	125
13.14	Reviewing Scenario Play.....	126
13.15	Transcript Windows	127
13.16	Reference Window	128
13.17	Help Window.....	130
14	Appendix H: Final Evaluation Feedback Form	131

List of Tables

Table 1. Major Scenario Development Tasks with Estimated Level of Effort by Labor Category	35
Table 2. Subjects' Levels of Experience.....	37
Table 3. Time Spent by Each Subject on Evaluation Tasks	37
Table 4. Subjects' Ratings of Effectiveness of Instructional Method.....	38
Table 5. Subjects' Ratings of Specific Aspects of Training System.....	38
Table 6. Proposed Task List with Capsule Descriptions.....	48

List of Figures

Figure 1. Top-Level METTLE Architecture.....	16
Figure 2. METTLE Server Functional Layering.	16
Figure 3. METTLE Briefing Screen.	20
Figure 4. Main METTLE Scenario Player Screen.	20
Figure 5. Simulated Patient Examination.	21
Figure 6. Patient Test Order Form with "Diagnostic Panels" Options.	21
Figure 7. Patient Chart – Tests Tab.....	22
Figure 8. Interaction with Another Character.	23
Figure 9. Tutor Hints and Feedback.....	23
Figure 10. Tutor Elicits and Refines Student's Current Differential Set.	24
Figure 11. Tutor Queries to Highlight Critical Findings for Anthrax Diagnosis.	24
Figure 12. METTLE Evaluation Ratings Distributions.	39
Figure 13. Venn Diagram of Student Utterance Spaces.	57
Figure 14. A Reasonable First Patient Interview Question, Badly Misspelled.....	68
Figure 15. A Follow-up Patient Interview Question with Multiple Meanings.	68
Figure 16. Simulated Patient Examination – The Eyes.....	69
Figure 17. Simulated Patient Examination – The Nose.	69
Figure 18. Asking for a <i>Hint</i>	70
Figure 19. Asking for More Explicit Direction on <i>What</i> to Do.	70
Figure 20. Asking for Explanation of <i>How</i> to Take the Recommended Action.	71
Figure 21. Asking for an Explanation of <i>Why</i> to do the Recommended Action.....	71
Figure 22. Pop-Up Menu of Treatment Categories.....	71
Figure 23. Patient Treatment Order Form with "Administer Fluids" Options.....	72
Figure 24. Patient Treatment Order Form with "Administer Antibiotics" Options.....	72
Figure 25. Tutor Reaction to Ill-Advised Student Action.....	73
Figure 26. Pop-Up Menu of Treatment Categories.....	73
Figure 27. Patient Test Order Form with "Diagnostic Panels" Options.	73
Figure 28. Tutor Reaction to Appropriate Student Action.....	74
Figure 29. Patient Chart – Admissions Tab.	74
Figure 30. Patient Chart – Complaint Tab.	74
Figure 31. Patient Chart – Exam Tab.....	75
Figure 32. Patient Chart – Vitals Tab.	75
Figure 33. Patient Chart – Tests Tab.....	76
Figure 34. Patient Chart – Treatments Tab.	76
Figure 35. Asking for Another Hint.....	77
Figure 36. Ordering Imagery Studies – Chest X-Ray	77
Figure 37. Positive Feedback on Chest X-Ray Order	77
Figure 38. Patient Chart – Results of Chest X-Ray.	78

Figure 39. Tutor Proactive Prompt Followed-Up by Student.	79
Figure 40. Student Reminded of Sick Cousin.	79
Figure 41. Student Calls Memorial ED and Tutor Launches Diagnostic Rationale Dialog.	80
Figure 42. Tutor Elicits and Refines Student’s Current Differential Set.	80
Figure 43. METTLE Reference Section, Conditions Tab with Focus on Brucellosis.	81
Figure 44. Link from METTLE References List to Authoritative Information Sources.	82
Figure 45. Some Tutor Critiques of Elements in Student’s Differential.	82
Figure 46. Transcript of Conversation with Memorial Hospital.	83
Figure 47. Final Diagnostic Rationale Discussion – Review of Key Patient Presentation Findings.	84
Figure 48. Final Diagnostic Rationale Discussion – Review of Key Microbiological Findings.	84
Figure 49. Final Diagnostic Rationale Discussion – Review of Key Imagery Findings.	85
Figure 50. Top Levels of METTLE Content Directory Structure.	86
Figure 51. Exercise-Specific Directory under htdocs.	87
Figure 52. Exercise-Specific Directory under servlets.	88
Figure 53. GRAIN Editor – Roles Tab.	89
Figure 54. GRAIN Editor – Courses Tab.	89
Figure 55. GRAIN Editor – Curriculum Tab.	90
Figure 56. Concept Editor – Conditions Tab.	90
Figure 57. GRAIN Editor – Exercises Tab.	90
Figure 58. GRAIN Editor – Students Tab.	91
Figure 59. Exercise Editor – Scenes Tab.	91
Figure 60. Exercise Editor – Actors Tab.	91
Figure 61. Exercise Editor – Briefings Tab.	92
Figure 62. Exercise Editor – Setups Tab.	92
Figure 63. Enact Behavior Spreadsheet Column Headers.	94
Figure 64. Behavior Spreadsheet with Simple Default Line Cues.	95
Figure 65. Behavior Spreadsheet with Simple Scenario-Specific Lines.	95
Figure 66. Behavior Spreadsheet with Complex Scenario-Specific Tutoring Line.	96
Figure 67. Generated spread_ File as Viewed in NotePad++ Text Editor.	97

1 Introduction

1.1 Purpose of the Research

DoD investment in technology for simulation-based training has proven useful across the combat branches, but has not been extended as vigorously to support areas such as medical services. Yet the teamwork and decision-making of medical professionals requires just as much training as is devoted to command of tactical units. Medical personnel too need extensive experience, expert coaching, and insight-producing after-action reviews in order to learn to see the factors that should affect their decisions and to prepare for likely follow-on consequences. The ability to provide such training in an anytime/anywhere distance-learning format is greatly enhanced by embedding instructional expertise in software and training scenarios.

Despite promising work on development of Intelligent Tutoring Systems (ITSs) for complex situation assessment, exploration, and response, it remains a research problem to efficiently and effectively build scenarios and tutoring that exploit what is known about good pedagogy for decision-making on relatively open-ended problems. Moving from individual training, to individualized training in the context of multi-role teams brings additional challenges and opportunities for scaling existing and planned technologies. *Our work focused on developing a framework, tools, and exemplary content all aimed at making it easier to build effective web-hosted simulation-based ITS scenarios for team-embedded complex decision-making skills in the medical environment.* We focused on training for medical response to low frequency but high impact Chemical, Biological, and Radiological (CBR) events, but the approach is applicable to a much wider range of medical diagnosis, treatment, and management situations. It also generalizes to non-medical applications where complex decision-making is naturally taught and critiqued in the context of scenarios requiring students to gather information and take action, in significant part by interacting with other agents, either verbally, or in other more stylized manners. We carried out our work in the context of civilian medicine to capitalize on greater access to subject matter expertise and stronger prospects for commercialization.

1.2 Brief Description of the Research

During this project we carried out five major tasks:

1. We researched existing CBR training materials and curricula, and gathered and analyzed extensive documentation on medical diagnosis, treatment, and systemic response to CBR incidents. We investigated aspects of military medical training in preparation for identifying transition opportunities in the military medical training establishment.
2. In support of system prototyping, we extended some of Stottler Henke's in-house tool suites, including the SimVentive™ toolkit for rapid simulation game construction (adding the ability to simulate continuous time, as well as several configurable user interface components), and the GristEdit knowledge base editing tools (greatly enhancing visualization and editing-display configurability), and experimented with novel authoring support techniques such as spreadsheet import/export. All of these efforts have borne fruit in various follow-on projects for the DoD.
3. We built and packaged for release a final version of the METTLE software that, in addition to meeting the goals of providing simulation-based medical decision-making training that embeds automated instruction, is also notable for its server-based logic and browser-based user interface relying only on standard mechanisms such as HTML, CSS, and JavaScript. METTLE *simulation* includes support for conversational agents. METTLE *instruction* includes proactive prompting, reactive feedback, and hinting/explanatory comments, as well as Socratic dialogs for decision rationale exploration. The varied Tutor capabilities help keep students moving forward and learning in the scenario, and provide observables and instruction at action and reasoning levels.

4. We developed a detailed training scenario focused on diagnosis of an index case of inhalation anthrax (the first case identified in a potential outbreak), assembled supporting materials (e.g. representative patient images and test results), and authored simulated agent behaviors, including tutor behaviors to support the scenario.
5. We carried out formative and final evaluations of the METTLE software and scenario. The formative evaluation provided impetus for mid-course corrections. The final evaluation provided a clearer picture of the strengths and weaknesses of the product at the end of the project.

1.3 Research Findings or Results

We highlight four major findings and results from this effort:

1. A set of software tools that support medical decision-making training scenarios, and that can be adapted for use in a far wider range of decision training applications that require mixed initiative dialog with simulated agents (including Socratic rationale exploration), and a simulated tutor capable of proactive prompting, reactive feedback, and hinting/explanatory comments.
2. A methodology, tools, and set of content assets that can be used to speed construction of medical training scenarios. The method lays out ten specific steps for course and scenario design and development. The tools support different types of authors with the different steps of the process. The existing content assets reduce the amount of from-scratch authoring required for each new scenario.
3. A proof-of-concept CBR training scenario that demonstrates the applicability of our technology for web-hosted simulation-based ITSs to medical decision making. The resulting scenario employs conversational text and forms based interaction, augmented by audio, graphics, and interactive images to make world events visible to the student, and to make student actions and understanding visible to the system.
4. Evaluation data from ten emergency physicians along eleven major criteria that yield average ratings of 4 on a five-point scale. In the first four *comparative* judgments, METTLE earned scores between 3.8 and 4.1, where 1.0 was “much less effective,” 3.0 was “as effective,” and 5.0 was “much more effective” than traditional approaches such as attending lectures or reading articles. In the other seven *absolute* judgments of specific system features, subjects reacted most favorably to the Tutoring capabilities. The METTLE Tutor’s ability to *support discussions about decision-making rationale*, and to *help students identify and address their knowledge and skill gaps* both earned ratings of 4.2 on a scale where 1.0 was “not effective,” 3.0 was “somewhat effective,” and 5.0 was “highly effective.”

2 Body

2.1 Problem, Solution, Innovations, and Objectives

Section 2.1 is adapted from the text of the project proposal, modified as necessary to reflect shifts in emphasis that developed over the life of the project. It is included here to provide context for understanding the Results reported in Section 2.2.

2.1.1 Problem Being Addressed

A well-functioning health-care system forms a key component of any society's infrastructure for self-maintenance, defense, and survival. In the present day, we see the need for improved medical skills ever more sharply. For instance, as the threat of mass casualties from unconventional weapons increases—both from overmatched conventional enemy forces on the battlefield and from terrorists intent on bringing their battles to the American homeland—medical professionals will play critical roles in recognizing and responding to such threats. Their knowledge, skills, and judgment, their ability to adapt to new situations and to work in dynamically formed teams with other medical professionals—these will largely determine the extent of the damage suffered when we face the reality of attacks that are no longer unthinkable.

Medical professionals will be the lynchpin in our response to Chemical, Biological, and Radiological (CBR) threats. Consider, for instance, the problems of dealing with an attack using biological agents—the so-called “poor man’s atom bomb.” First of all, the attack itself would probably be clandestine, and with current technology is likely to be undetectable. Second, the incubation period before symptoms emerge might range from hours to days to weeks, or even months. Third, the original pattern of dispersion is highly unpredictable, depending on vagaries of wind and weather. Fourth, dispersion of pathogens in weapons form may lead to infection through atypical portals which means diagnosis may be complicated by atypical symptoms. Fifth, an enemy may further deliberately confuse diagnosis by using combinations of infectious agents. These complications of stealth, delay, unpredictability, atypicality, and misdirection combine to compound the basic problem of communicability: infected people can unknowingly infect others. In particular, without appropriate precautions, those most needed to manage an outbreak and provide treatment—the members of the medical corps—may become infected and incapacitated. Add to all this the potential for panic when knowledge of the attack become public, and you clearly have the potential for a problem of overwhelming proportions.

Medical personnel are rightly recognized as prototypical “professionals”—individuals who devote themselves to developing and expanding specialized knowledge and skills that enable them to assess and respond to complex and highly variable problems. The problems they face every day—and the problems they will face in the event of CBR attack—are not of the sort that yield to simple application of doctrine and procedure, but the kind that require informed professional judgment and (often ad-hoc) teamwork. An insightful review of early efforts by the U.S. government to take seriously the problem of planning for domestic chemical or biological attack (Seiple, 1997) describes the experience of standing up the Marines’ CBIRF, and preparing for the Atlanta Olympic Games. In his analysis, the most important issues had to do with coordination of ad-hoc teams required to assemble all the varied expertise needed to address a potential problem. Also—not surprisingly—military units were credited with being the repositories of critical expertise unduplicated in the civilian world. Again, the point is that we must train medical professionals to cope with large scale medical emergencies, and that such coping is largely about making good decisions and functioning effectively as part of an ad-hoc multi-disciplinary team.

While it is fortunate that our real-world experience with these scenarios is quite limited, that only makes the planning and training problems harder. What is not well practiced is generally not well performed. Processes that have only been envisioned hypothetically have not had the kinks worked out of them. Too much is at stake here not to have a well designed, well practiced, and broadly disseminated basis for response. Doctrine in manuals and plans on paper are a good start. But all medical personnel

must be able to turn prior planning into appropriate decisions and actions when the time comes, as it seems inevitable that it must.

Within the U.S. military there is a tremendous ongoing commitment to ensuring our forces are the best trained in the world. Training and training development for all branches and all specialties is generally ongoing. Training objectives span a range of complexity from component skills, to operational skills, through higher-level decision-making, and on to effective team operations. Training mechanisms run the gamut from self-study materials, to distance learning, live courses, individual and group simulations, and ultimately live exercises of varying scales. Computer-based simulations have constituted a growing segment of the training spectrum, as massive investment and consequent technological advances have raised capabilities and lowered costs. Simulations are now widely used across the services to provide increased opportunities for practice and training at decreased cost.

However, the investment in technology for simulation-based training that has proven so useful in the combat branches, has not always been extended to key support areas such as medical services. And yet the teamwork and decision-making of medical professionals requires just as much training as is devoted to command of tactical units. They too need extensive experience, expert coaching, and insight-producing after-action reviews in order to learn to see the factors that should affect their decisions and to prepare for the likely follow-on consequences of their decisions, as well as for the potential of further assaults by a hostile adversary.

The military has also largely recognized that simulations, by themselves, are not particularly useful as training; it is the coupling of simulators, with appropriately crafted scenarios, and expert coaching and feedback that provide the greatest benefit. However, the need for expert supervision drives up the cost and limits the availability of effective training. In response, much work has been devoted to coupling individually responsive automated instructional capabilities with simulations. The goal is to continue to improve the cost/benefit equation by minimizing the need for human observer/controllers in simulation-based training. Intelligent Tutoring Systems (ITSs) are an emerging technology that puts a simulated instructor in the computer box along with the simulated world (Ong & Ramachandran, 2000). The level of ambition and proven capability for ITSs has also been increasing, and research is pushing practical ITS tools from the level of straightforward procedural tasks (e.g. Munro & Pizzini, 1995) or closed-world formal reasoning tasks (Anderson, et al, 1985) to more open-ended analysis and decision tasks such as tactical decision-making (Domeshek, Holman, & Ross, 2002).

Despite progress, building affordable and effective ITSs for complex decision-making and team-coordination tasks remains a research problem requiring unique combinations of specialized expertise. For instance, such an ITS calls for a careful and novel combination of simulation components and training scenario design. We cannot simply let a simulator run free, as the simulated world could easily get into states that are either not instructionally relevant, or where the system is not prepared to critique and tutor. Since there is no validated algorithmic approach to carrying out in all situations the jobs the system is intended to teach, we cannot apply the common ITS approach of building a fully competent expert system, and then using that system to structure an “overlay” model of student competence and generate arbitrary problems. Instead, we have to characterize islands where general principles drive team members’ behaviors, and supplement those with scripts and contextually-bound assessments designed to teach specific points in specific scenarios. Problems with coverage actually exist on the simulation side as well, since no one validated simulator can accurately generate all world states that might be relevant to our target training audience. As noted in (Smith, 2003), there are many simulators already in existence that could be relevant to this training problem. However, we must be prepared to fall back on scripted events to maintain focus on pedagogically useful situations and assessable sequences of actions. Finally, designing scenarios that exploit what is known about good pedagogy at this level, and efficiently and effectively building automated behaviors and instruction remains a challenge. Moving from individual training, to individualized training in the context of multi-role teams brings further challenges, as well as opportunities for scaling existing and planned technologies.

Since initially framing this broad problem statement in response to the original research solicitation, we have substantially focused our work towards development of scenario-based conversational ITSs for emergency medical decision-making and team-coordination skills in the context of hospital-based teams. In that context, key issues include (a) diagnosis of early cases potentially leading to discovery of the existence or nature of a CBR event, (b) treatment of individual casualties of a CBR event, and (c) more systemic responses to the recognition of a CBR event directly affecting future medical operations.

2.1.2 Overview of Solution

The goal of this project is to develop training that can provide medical professionals (military and civilian) with extensive simulated practice and coaching on decision-making required to deal with medical emergencies (e.g. CBR attacks). Further, the goal is to make this training widely, easily, and cheaply available to the large number of professionals who might find themselves responsible for contributing in different roles to a coordinated response to such a situation.

Given the current state of technology, developing a web-hosted simulation-based ITS is among the most promising approaches to this problem. From a centrally administered server (or distributed family of servers as needed), users can cheaply and easily access such training, wherever they may be and whenever they have time. Likewise new insights regarding possible threats or approved response doctrine can drive creation or modification of training scenarios, and such updates can be deployed on the servers, becoming immediately available to the whole student community.

To ensure the widest accessibility of such training, the components that run on the student's client machine (in their web browser), should avoid making unnecessary assumptions about system capabilities. In particular, they should adhere to the most common (web) standards, minimize demands on network bandwidth, refrain from placing undue loads on client memory or processing, and function well with standard input/output capabilities. To enable effective training whenever an individual student has time, scenarios should be populated with simulated agents playing the roles of other team members. Variant of scenarios can be constructed to focus on the decisions and interactions appropriate to particular roles in the overall medical response (e.g. scene sampling and analysis, front-line triage, decontamination, intake and initial processing, long-term care, resource load planning, large-scale logistics, etc).

No matter the role, the focus on complex decision-making in multi-character contexts dictates that student interaction with the system be characterized in large part by extensive dialog—both with simulated teammates and with the embedded tutor. When decision-making is the focal task in a team environment, the interactions among team members are mostly about information exchange, task coordination, and responsibility allocation—all requiring extensive communication. When contextual judgment is the focal learning objective in a tutoring environment, the interactions between student and tutor most effectively dwell on decision rationale—requiring the teasing out of relevant factors through ongoing (often Socratic) dialog.

We built a Medical Emergency Team Tutored Learning Environment (METTLE)—an ITS runtime, authoring, and distribution environment that satisfies the constraints above. METTLE exploits, extends, and refines a novel combination of technologies Stottler Henke has developed in the context of several ITSs for professional-level decision-support domains. We use the acronym SPIRIT to summarize the key components of our approach: *Scenarios, Principles, Issues, Roles, Interactions, and Tools*:

- **Scenarios:** Scenarios are constrained simulated experiences designed to raise particular issues and teach particular principles. The constraint on the experience comes from the initial simulation conditions, from consequent (conditionally) scripted events, and from biases in the responses of simulated agents. Within these limitations, an underlying simulator (or cluster of partial simulators) may calculate the world's evolution, perturbed by student actions. For instance, a scenario might be designed to emphasize (among other issues) the consequences of appropriate or inappropriate decontamination or isolation regimens following a CBR attack.

- **Principles:** Principles are the main points a system like METTLE is designed to teach. Principles may be generalizations, often at a level that might be characterized as “control knowledge” for application of a skill (e.g. in case of suspected biological attack, once you have identified a likely pathogen, be sure to check again for other possible agents that might have been combined with the one you have identified), or they may be more specific contextually bound knowledge (e.g. when dealing with inhalation anthrax, expect an incubation period of 1-7 days, but allow for the possibility of a period of up to 60 days).
- **Issues:** Issues characterize key choice-points in the decision-making required by a scenario. As such, they often serve as organizers of principles—the knowledge, skills, and control required to reasonably address the issue. Example issues for METTLE might include when to decide a potential patient is beyond help, how long to maintain a potential victim in isolation, or whether to recommend prophylactic treatment related to a suspected threat.
- **Roles:** METTLE is intended to train individuals in tasks where they must act as part of a team. We assume the team is not homogeneous—that is, different team members have different roles, and thus different responsibilities (different issues, subject to resolution by different principles). Example roles include first-responder, front-line care-giver, consulting specialist, long-term care-giver, incident manager, resource planner, etc.
- **Interactions:** In METTLE’s domain of application, individual medical personnel cannot do their jobs without interacting with other characters. As noted, we developed computer-simulated agents to play patients and team members, for convenience, flexibility, and focus. Those agents must support reasonably realistic interaction be it face-to-face conversation or phone contacts.
- **Tools:** The individual medical personnel also cannot do their jobs without relying on some set of tools. METTLE’s user interface provides a simulated patient chart, various order forms, and a visible patient to support examinations. A combination of dialog (interactions), multimedia presentations, and interactive visualizations is, in general, required.

2.1.3 Innovations

METTLE’s core innovation is the extension of ITS capabilities to decision-making at the challenging level of professional judgment in simulated team contexts on open-ended problems. METTLE is not aimed at *procedural* training; it focuses on reasoning and decision-making. Further, METTLE is not just about *individual* reasoning and decision-making; it also addresses learning to make judgments and act as part of a team. Finally, METTLE is not about decision-making on *formal or easily formalized* problems; it provides training to human experts on a class of problems for which it is not yet possible to build a computer-based solution (e.g. expert system). The technology that enables this innovation includes the combination and extension of several Stottler Henke specialties:

- **Scenario-scoped principle-driven simulation-based interaction and assessment.** Medical personnel need extensive practice with the decision and teamwork skills required for successful emergency response, and simulation is the most cost-effective, readily accessible way to provide that practice. But since, for this kind of large problem, unconstrained simulation is neither practical nor pedagogically useful, a scenario-based, principle-driven approach is necessary. In order to put the instructor in the box along with the simulation, artificial intelligence (AI) based techniques for automating student performance assessment in the context of simulated scenarios are also required. In order to put the equivalent of a good professional-level mentor in the box, additional techniques for Socratic tutoring are required.
- **Role-based simulated-agent and dialog-interaction capabilities.** For this application, we need to put not only the instructor, but also the student’s teammates in the box. This requires an agent simulation capability that mates with the underlying scenario simulation machinery. Further, for this application, the dialog-capable simulated agents represent a variety of roles, each with their own goals, responsibilities, knowledge, and capabilities within the simulated world, and some of

these simulated roles can potentially be taken by a student in a variant of the same scenario. The authoring infrastructure needs to support multi-role authoring and capitalize on the potential for reusing the same situation to train different role-fillers.

- **Rich simulation interaction delivered through standards-based web client.** During the course of the project our user interface goals shifted somewhat, leading us to invest more effort in achieving standards-compliant web-delivery, and less in configurability. This was a consequence of trying to push the envelope on *deployability* (a stated project goal, reinforced by sensitivity to commercialization prospects) while maintaining usability and richness of experience. The resulting ability to deliver a simulation-based ITS with conversational, graphical, and forms-based interaction, while relying only on HTML, CSS, JavaScript, and standard media formats (e.g. jpg, wav) is a unique and useful capability that greatly eases system access.

2.1.4 Phase II Technical Objectives

The technical objectives for this Phase II project included the following:

1. **Develop a simulation role-play environment for medical response to CBR emergencies.** Our stated goals with regard to this objective were to establish an integrated simulation environment appropriate to the training problem we were addressing, while improving language-based interaction, and ensuring the system was flexible and configurable. Implicit in this objective was consistency with later objectives emphasizing ease of specifying scenario scripts, and efficacy in instruction. Though the project made some early and enduring contributions to the last goal, we eventually shifted our emphasis from rapid configuration of simulation and visualization components to web delivery of training simulations in pursuit of a more pragmatic (and commercializable) approach to deployability. With that caveat, the results reported below address this objective, including development of a simple, consistent, and extensible behavior interpretation framework, including special support for two kinds of language-based interaction: student-initiative and agent-initiative dialogs, packaged for delivery through unaugmented COTS web browsers.
2. **Develop automated tutoring for students in the above role-play environment.** Our stated goals with regard to this objective were to develop more general underlying ITS techniques to enable sensitivity to pedagogically distinct situations and production of pedagogically appropriate responses. Again, this objective required consistency with later objectives emphasizing ease of specifying instructional behaviors and instructional efficacy. In response we continued to exploit and extend the general behavior interpretation framework referenced above, designing mechanisms whereby tutor behaviors can take advantage of the same test and action languages (affording them broad scope as simulated agents in their own right), while being specified largely as straightforward annotations on base character behaviors. We also exploited the agent-initiative dialog capability to build Socratic dialogs that carry a significant part of the instructional burden. In the diagnostic context, a common instructional goal is to review student differentials to explore and prompt their thinking; we developed conventions for structuring these key discussions.
3. **Develop authoring tools to speed construction both of scenarios and of tutorial instruction.** Our stated goals with regard to this objective called out several different kinds of authoring tools to be developed. As described below, we have addressed METTLE's authoring needs through a mix of techniques that layer nicely: (a) extensive use of standard web and media formats that can be produced by COTS tools, (b) a bedrock level of human-readable data formats for essentially all system data that can be edited using COTS text editors, (c) exploitation of COTS spreadsheet applications over a pre-specified format capable of capturing the vast majority of agent behaviors, and (d) a set of custom web-based tools for specifying most course content. Though gaps remain in our tool set, most kinds of content creation can be accomplished with ease and speed.

4. **Demonstrate system efficacy in pilot evaluation studies.** As of this Final Report we have data that suggests: (a) the system is distributable and usable by physicians with very little assistance, (b) physicians generally find the system both engaging and appropriate, and (c) on average they believe it to be pedagogically effective on several dimensions, reported in more detail below.

2.2 Project Results

This Section constitutes the centerpiece of this report. Our major results over the lifetime of the METTLE Phase II project fall into eight categories, each discussed in a subsection below (and many elaborated in greater detail in appendices): (1) modules and concepts contributed to existing frameworks, (2) final system architecture, (3) final system runtime, (4) sample scenario, (5) authoring capabilities, (6) authoring methodology, (7) evaluation results, and (8) cost/benefit analysis.

2.2.1 Tool Framework Modules and Concepts

As laid out in our proposal, Phase II development work on METTLE was arranged as three cycles of design, implementation, and evaluation. Each cycle was an experiment, emphasizing particular target system features, and building on some available infrastructure. The first cycle focused on elaboration of the METTLE architecture, experiments with speech I/O, and early work on authoring tools; it built on the Phase I prototype code base, plus commodity desktop database tools. The second cycle focused on simulated dialog mechanisms (both student and system initiative), runtime user interface elaboration, and a more flexible approach to authoring; it built on and contributed to Stottler Henke's SimVentive and GRIST toolkits, and explored ways to exploit commodity spreadsheet applications for scenario content authoring. The third and final cycle focused on improved system performance, web-delivery, and easier integrated authoring of tutor behaviors with simulated character behaviors; it built on the extensible open source MzScheme web-server, and integrated an elaborated version of the spreadsheet authoring tools.

While the bulk of this report focuses on the final version of the system, in this subsection we highlight some METTLE-developed concepts and modules that constitute durable contributions to other Stottler Henke tools and frameworks—contributions that are significant in that they have found use in other projects or products, primarily supported DoD missions.

2.2.1.1 SimVentive Extensions

During the second cycle of development, which received a significant boost by leveraging the SimVentive tool base, METTLE in turn funded development of SimVentive extensions, including real-time modeling and several UI widgets. These extensions have since become part of the standard SimVentive product, and have been used in development of other ITSs.

Prior to the METTLE experiment, SimVentive had been limited to supporting turn-based simulations. METTLE extended SimVentive to support real-time simulations. To complement this world modeling enhancement, METTLE also introduced time control widgets—user-visible methods to pause, stop, speed, and slow the passage of time. METTLE also introduced a novel UI widgets for incrementally collecting and displaying nicely formatted transcripts of ongoing scenario play. Finally, METTLE built a UI widget for displaying interactive sets of graphics with modifiable hotspot regions. Along the way, METTLE helped drive requirements and refinements to SimVentive, such as improving the ability for multiple authors to coordinate their work on a common scenario, and to support more complex UI layouts and editing. METTLE also drove integration of SimVentive with the GRIST modeling framework (see immediately below).

2.2.1.2 GristEdit Extensions

GRIST is Stottler Henke's in-house knowledge representation and reasoning framework, originally developed in the context of the ComMentor ITS, and since refined and enhanced through incorporation into a major multi-user server-based application. Our second cycle of METTLE development leveraged GRIST to represent not only all aspects of the simulated world, but also many aspects of agent behavior

specifications. This eventually led to a level of integration between SimVentive and GRIST that has led to GRIST shipping as the standard world modeling formalism in SimVentive.

The GRIST representation of both world and behavior provided an opportunity to address METTLE authoring needs using extensions to the standard tools for visualizing and editing GRIST structures: the GristEdit suite. METTLE funded extensions to GristEdit that produced a customizable multi-editor capability (sometimes called CustomEdit), where a tabbed interface allowed easy switching between tailored views of subsets of an overall GRIST knowledge base. These views emphasized list/tree views of collections and property-sheet views of individuals, but also included matrix views of inter-object relationships. These capabilities were used to elaborate METTLE's space of simulated patient behaviors, and provided a way to create and visualize relationships, e.g. between symptoms and diseases, or diseases and treatments.

These GristEdit enhancements have since proven key to the rapid prototyping of totally different class of application: a scheduling and group collaboration tool for managing modernization of U.S. Navy ships benefited from the ability to rapidly configure tailored views of elements from a GRIST model describing Ships, Warfare-Systems, Upgrade-Processes, Ship-Configurations, and so on.

2.2.1.3 Spreadsheet-Based Authoring

The final durable product of our second development cycle is somewhat conceptual (as compared to the specific code artifacts described above). As evidenced by our attempts to adapt commodity database tools for authoring during the first development cycle, and our use of other in-house tools during the second cycle, we have a long-standing bias towards exploring ways to leverage existing tools for novel purposes. This potentially has benefits in terms of tool quality, robustness, and familiarity, as well as efficiency in development. The novel concept we developed here was (1) factoring the authoring problem into relatively simple/conventional pieces versus more complex/custom pieces, (2) developing straightforward mappings of the simple pieces into standard tabular formats, (3) allowing authors to create, review, and maintain those simple tables using conventional spreadsheet applications, and (4) within those limitations, finding ways to support the author in creating valid behavior specifications (e.g. pre-generating parts of the tables, exploiting spreadsheet validation capabilities, etc.).

Despite the CustomEdit extensions described above, it became clear (based on our own judgments and more especially, those of our consultants) that the resulting authoring tool suite was still not as easy to use as we desired. In particular, it is very difficult to match the level of maturity and capability inherent in a spreadsheet: e.g. ability to robustly handle, gracefully display, scroll-through, and search large amounts of (lightly formatted) data, support for individual element and group cut/paste, ability to easily print out nicely formatted subsets of data for off-line review, and even the ability to email to essentially any interested party and expect them to be able to open and inspect the data.

As the underlying runtime machinery matured, and as our initial scenario behavior scripts developed, we were able to identify the most common use-patterns and develop spreadsheet formats, plus import and export mechanisms to enable authors to do much of their work using these more familiar tools. As described further below, this basic approach has been carried over to the third and final version of METTLE. The same approach has also been applied successfully in several other ITS development efforts within Stottler Henke.

2.2.2 Final System Architecture

The final METTLE system architecture reflects many lessons learned across the several development cycles. We believe it is a robust, flexible, and extensible architecture. One piece of evidence is that it is already being applied and adapted to support a novel ITS development effort that is simultaneously exploiting the basic multi-agent conversational simulation, while adding new capabilities that integrate lessons-learned access with scenario simulation and role-play.

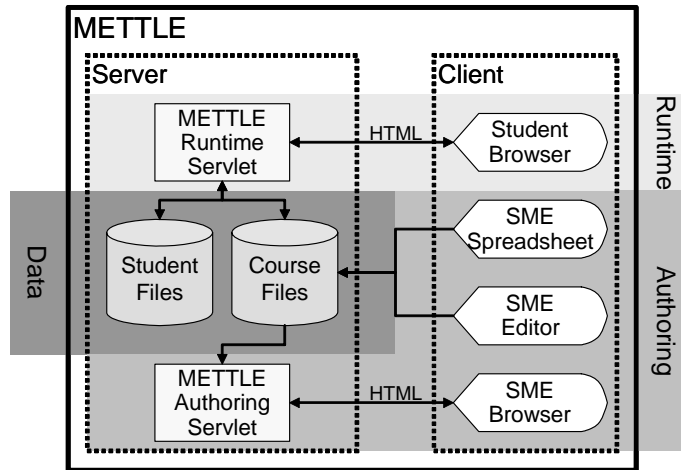


Figure 1. Top-Level METTLE Architecture.

Figure 1 shows METTLE’s top level architecture, emphasizing the distinction between **Server** and **Client** on the one hand, and between **Runtime** (see 2.2.3) and **Authoring** (see 2.2.5) on the other. As suggested earlier, the Server is an augmented web-server; the Client is a web browser. We also note that Subject Matter Experts (SMEs) serving as authors may use more conventional file-based tools (e.g. spreadsheet applications, or text editors) to prepare some of the data that drives the Student experience.

Figure 2 focuses on the METTLE Server, describing its internal structure by focusing on functional layering. The top layer (layer 8) corresponds to the METTLE Runtime and Authoring Servlets shown in Figure 1. All other layers of software exist to support that end user functionality. Detailed discussion of the most important and novel layers (i.e. from 7 down to 4) is deferred to Appendix B. Here we briefly summarize the purpose of each layer.

8. METTLE	Runtime Servlet	Authoring Servlet
7. Enact	Enact Structures	Enact Interpreter
6. Discuss	Discuss Structures	Discuss Interpreter
5. GRAIN	GRAIN Structures	GRAIN Interpreter
4. Interpreter	Test Interpreter	Action Interpreter
3. Concept	Concept Structures	Concept HTML Forms
2. Registry	Registry Structures	Editable HTML Widgets
1. Web	Servlet Framework	Supporting JavaScript
0. Utility	List Processing	Text Processing

Figure 2. METTLE Server Functional Layering.

0. **The Utility layer** provides very basic functionality, primarily related to list structure and text processing, most of which is too low-level and commonplace to be worth going into in any detail here. However, we note two key text processing capabilities that support bag-of-words text matching: *Stemming*, the removal of prefixes and suffixes to increase the chances of matching common roots is effected by an implementation of the standard Porter stemming algorithm (Porter, 1980); *Stop-Word Removal* eliminates common low-content function words such as “and” “or” “is”, etc. to cut down on meaningless term matches.

1. **The Web layer** provides a convenient framework for defining complex web applications (servlets). In this framework, an application is defined as a set of pages, each page is defined as a fixed layout of panes, and pages and pane have associated methods for generating and combining HTML, CSS, and JavaScript. Session management, page transition management, and associating HTTP requests to application actions are all conveniently supported. An accumulating set of JavaScript libraries (many borrowed and adapted from around the web) allow more advanced browser-based behaviors, such as pop-up menus and rich text (HTML) editing.
2. **The Registry layer** provides a simple infrastructure for creating, editing, and persisting lists and trees of named data objects. It is used to implement the Concept layer and the GRAIN layer. Its core capabilities include tracking the existence, name, and relative positioning of registered data objects. Its companion set of editable HTML widgets make it possible to build interactive web-based editors for these registered objects. Thus this layer also supports the Concept editor and GRAIN editor aspects of the METTLE Editor servlet.
3. **The Concept layer** provides a very simple hierarchically structured data store with slot/filler capabilities. It organizes its concept hierarchies into disjoint spaces and allows for linkages within and across spaces. Some of the major concept spaces used by METTLE include *conditions*, *findings*, *tests*, and *treatments*. The underlying Registry layer provides editing and persistence capabilities for concepts.
4. **The Interpreter layer** is a core technology that enables and unifies many of the capabilities of the overall system. The Interpreter is an extensible object-oriented framework for defining and composing small instruction languages. The resulting composite language can be used to create test and action expressions. Tests produce true/false values. Actions produce effects within the system. Together they can be used to define agent behaviors.
5. **The GRAIN layer** is a first implementation of a General Runtime and Authoring for Instruction, intended to provide simple Learning Management System (LMS) capabilities in a way that works well with an ITS. It allows simple representations for *courses* and *curricula*, *instruction* and *exercises*, *students* and the accumulating *records* of their experience with the system. METTLE scenarios are GRAIN exercises, and are able to use GRAIN commands to update a student model.
6. **The Discuss layer** is an implementation of the design for agent-initiative dialog control first described in the Year 1 Report as the Dialog Package. As compared to the simpler stimulus-response behavior-based dialog capability enabled by more direct use of the Interpreter layer, it provides a more elaborate control mechanism that allows a simulated character to drive and shape an extended dialog. Critical to support of METTLE's target decision-making training, such dialogs will often be *Socratic* explorations of the rationale underlying student actions.
7. **The Enact layer** is an implementation of the design first described in the Year 1 Report as the "METTLE Machine"—an abstraction capturing much of the structure of METTLE-style ITS scenarios. Enact is responsible for orchestrating the flow and interaction of *scenarios*, *scenes*, *actors*, *states*, and agent (including tutor) *behaviors*. At its core, it supports a kind of **contextualized stimulus-response behavior modeling**, specifically including agent responses to student-initiative dialog cues.

2.2.3 System Runtime

The METTLE runtime is a web application structured as a set of pop-up web pages, each with a fixed multi-pane layout. A brief overview of METTLE scenario play is presented in the next section, while a more detailed graphical tour through the runtime interaction following the thread of a particular scenario run is provided in Appendix C; the system Help pages are presented in Appendix G. Together those give a complete picture of what the runtime system looks like and how it functions. Here we just give a quick overview of the sixteen pages that comprise the final application:

1. **Login:** The login page provides a way for the student to identify themselves to the system; this is important as it enable the tracking of individual student actions and the compilation of student records. This page also provides options to transition to the **New User** page and to the **Credits** page. On successful login, the student is taken to the **Chooser** page.
2. **New User:** The New User page provides a way for new users to identify themselves to the system and create new accounts. On successful creation of a new user account, the student is taken to the **Chooser** page.
3. **Credits:** The Credits page provides information on who built and funded METTLE. On leaving this page, the student is returned to the **Login** page.
4. **Chooser:** The Chooser page provides a listing of available scenarios, a preview/synopsis of a currently selected scenario, and the ability to start running a chosen scenario. On launch of a scenario, the student is generally taken to the **Briefing** page.
5. **Briefing:** The Briefing page displays normal HTML intended to provide an introduction either to an entire scenario, or to a newly initiated scene within a scenario. A scenario introduction ought to describe the place and time at which the scenario is taking place, the Student's role and goal, and other relevant background information such as admissions information on a new patient. When the student has finished absorbing the briefing, they can click the 'Play Scenario' button and move to the **Player** page.
6. **Player:** The main scenario Player page is the center of the simulation role-play. The left half of the page is given over to display and interaction with simulated actors in the scenario. The right half of the page is given over to display and interaction with the simulated Tutor. Most of the remaining pages are pop-ups that can be invoked from the Player page.
7. **Chart:** The Chart page appears when the student clicks the 'Chart' button for a simulated patient. It displays the multi-tabbed patient chart for the current patient. Some tabs of the patient chart display relatively static system-generated data (e.g. the admissions cover-sheet). Most display more dynamic data that reflects actions the student takes in the simulation. The most important of these tabs is the one that displays test results.
8. **Examination:** The Examination page appears when the student clicks the 'Examine' button for a simulated patient. It provides a means for performing a graphical examination of the patient. The left half of the page displays side-by-side front and back full-body views, each with a large number of clickable hot-spot regions. The right half displays a detailed image, depending on what region of the overall patient view was most recently clicked. At the bottom of the page, highlighted findings from the most recent examination click are displayed. These are also generally automatically recorded in the appropriate section of the patient chart.
9. **Test:** The Test page appears when the student clicks the 'Test' button for a simulated patient, and chooses one of the options from the associated pop-up menu items. The page is populated with a particular test order form determined by which menu item the student chose. By selecting checkbox items on the form, and then clicking the 'Submit' button, the student can order a wide range of tests to be run on the patient.
10. **Treatment:** The Treatment page appears when the student clicks the 'Treat' button for a simulated patient, and chooses one of the options from the associated pop-up menu items. The page is populated with a particular treatment order form determined by which menu item the student chose. By selecting checkbox items on the form, and then clicking the 'Submit' button, the student can order a wide range of treatments for the patient. Logically, this page (and the two following) probably ought to share the same pop-up window (and implementation) as the prior **Test** page, but a desire for custom sizing led to this slight proliferation of extra pages).

11. **Management:** Just like the **Treatment** page, only for order forms that offer patient management options (e.g. settling on a diagnosis, making a referral, etc) that are not precisely treatment decisions.
12. **Response:** Just like the **Management** page, only for order forms that offer options having more to do with the hospital status than with patient treatment or management.
13. **Reference:** The Reference page allows browsing through collections of key domain concepts and associated links to related information, available either from authoritative sources on the web (e.g. CDC) or from documents stored on the METTLE server.
14. **Help:** The Help page allows browsing through a network of HTML pages that describe how the METTLE system operates.
15. **Transcript:** The Transcript page can be popped up by the user to review what has happened during the play of a scenario. Transcripts can be produced either for the entire scenario interaction, or on a per-simulated-actor basis (e.g. just the history of the Student's interaction with a simulated patient).
16. **Scorecard:** The Scorecard page displays a summary of the student's experience and performance with various curriculum points. It provides a history of the exercises they have run (sorted from most recent to least recent), and by clicking on any one of them they can see the pieces of the curriculum tree that were exercised during that scenario, and how many times they succeeded or failed in the exercise of each such point.

2.2.4 Sample Scenario

The scenario we developed during our Phase II work asks a student to play the role of Emergency Department (ED) physician, and to attend to a patient who presents with intense flu-like symptoms. This patient is actually suffering from inhalation anthrax; it is a large part of the student's job to uncover and interpret the evidence that will allow them to reach that conclusion. A detailed presentation of many steps in a typical run through the scenario appears in Appendix C. Here we provide a small excerpt from that larger run to give a flavor of the resulting interaction style. Actual play typically takes about an hour.

As shown in Figure 3 the scenario begins with a briefing page that defines the student's role and introduces them to a patient, Ryan Smith. The full briefing (much of which is scrolled off the bottom of Figure 3, but is included in Appendix C) includes additional intake information, and concludes with basic instructions on how to proceed. The main player screen is shown in Figure 4. The left half provides the context for interacting with characters in the simulated world; the right half is for interacting with the automated Tutor (and accessing miscellaneous system control functions). Figure 4 illustrates text-based interaction with the simulated patient. Near the top is the result of a just-completed exchange. At the bottom the student has typed in the next thing they want to say, and clicked 'Try;' in response the system has echoed back a list of checkbox items that represent it's best interpretation of what the student might have meant. In this case, it turns out the student meant both of the top-two system choices, so the student can check both of them and then click 'Say' to complete their utterance. COTS speech input technology interoperates well with standard browsers, and the system can play recorded sound files of a character's speech, so speech based I/O (though with the 'Try'/'Say' confirmation step) is also possible.

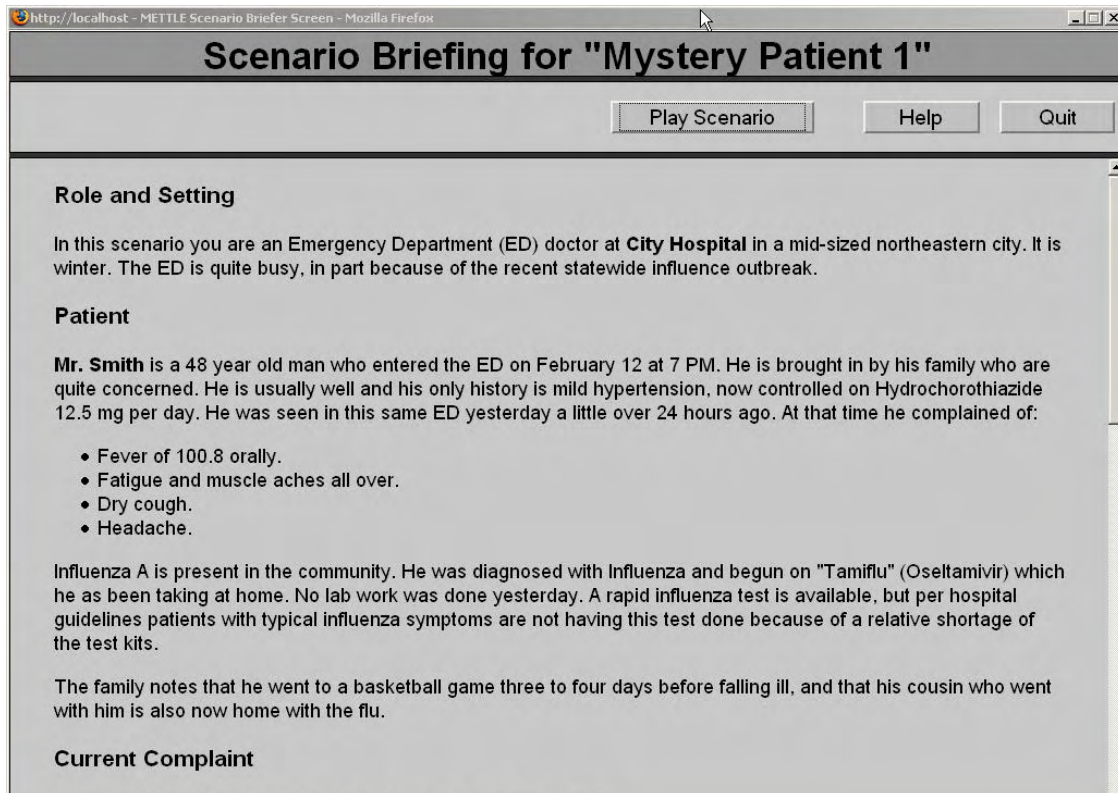


Figure 3. METTLE Briefing Screen.

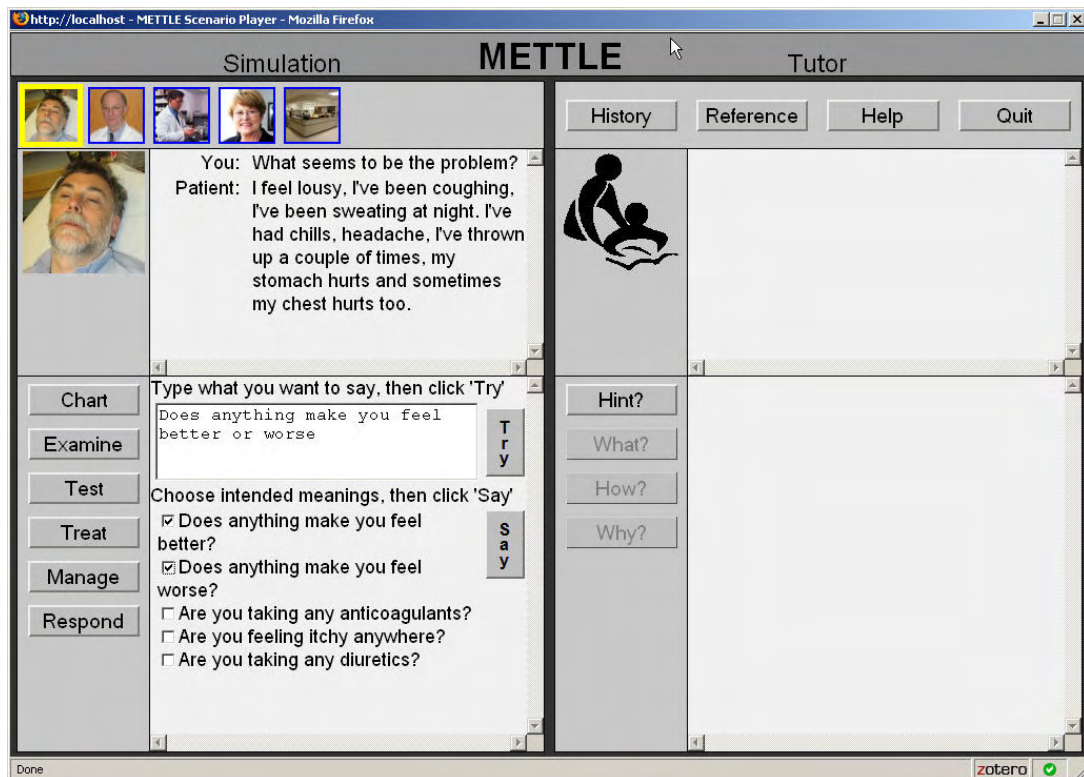


Figure 4. Main METTLE Scenario Player Screen.

Language-based interaction is the default mode provided by the Player screen. But the buttons in the left margin under the picture of the patient open windows that provide other ways to interact. Figure 5 shows the window that pops up when the student clicks on ‘Examine.’ The left half of the Examination screen shows a fixed image of the entire patient, front and back. The right half is updated with a detailed view depending on which hot-region the student clicks in the overview. The bottom also updates with a summary of any findings produced by examining that part of the body.

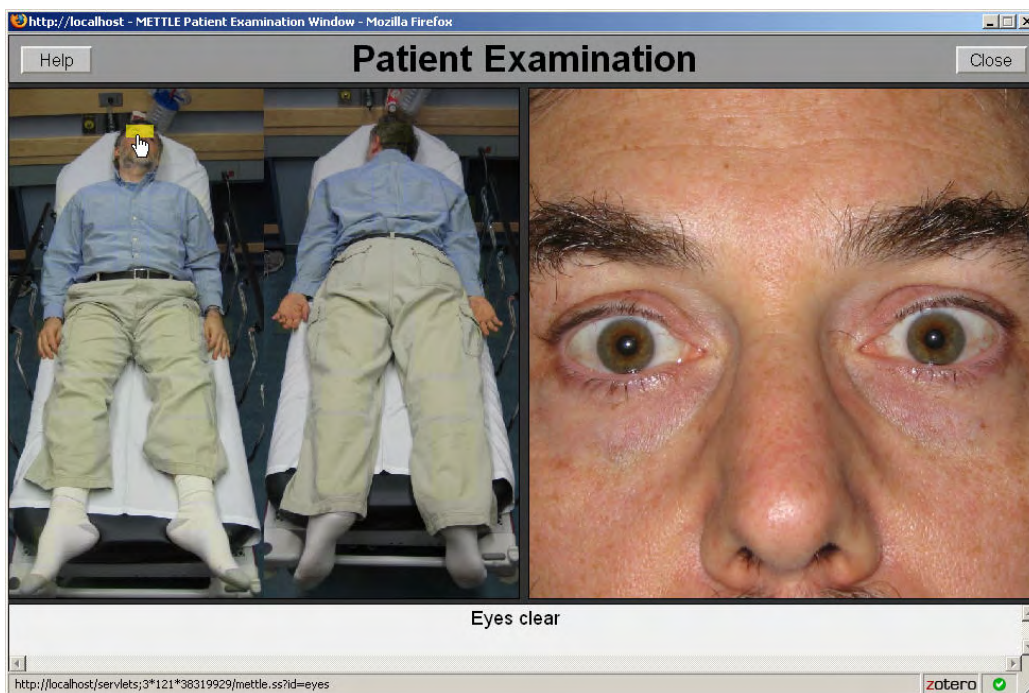


Figure 5. Simulated Patient Examination.

The other buttons beneath ‘Examine’ reveal menus that allow for choices of pop-up order form windows. For instance Figure 6 shows the order form for Diagnostic Panels. The results of all patient interactions—interview, examination, tests, and treatments—are recorded in the patient Chart. Figure 7 shows the results of tests orders in Figure 6.

Diagnostic Panels	
<input checked="" type="checkbox"/>	Arterial Blood Gases Test
<input checked="" type="checkbox"/>	Basic Metabolic Profile
<input type="checkbox"/>	Comprehensive Metabolic Profile
<input type="checkbox"/>	Electrolytes Panel
<input type="checkbox"/>	Electrolytes Urine
<input checked="" type="checkbox"/>	Urinalysis Complete
<input type="checkbox"/>	Urinalysis Dipstick
<input type="checkbox"/>	Hepatic Function Test
<input type="checkbox"/>	Serum Drug Screen 6 Drugs
<input type="checkbox"/>	Urine Drug Screen 2 Drugs
<input type="checkbox"/>	Urine Drug Screen 3 Drugs
<input type="checkbox"/>	Urine Drug Screen 7 Drugs

Figure 6. Patient Test Order Form with “Diagnostic Panels” Options.

Patient Chart			
		Tests	
Student	02/12/2007-19:09:51	Order Urinalysis Complete	
		Color	yellow yellow
		Clarity	Clear Clear
		GLUCOSE, URINE	NEGATIVE NEGATIVE
		BILIRUBIN, URINE	NEGATIVE NEGATIVE
		KETONE, URINE	NEGATIVE NEGATIVE
		SPEC GRAV URINE	1.020 1.002-1.035
		OCC BLD URINE	NEGATIVE NEGATIVE
		PH URINE	5.0 5.0-8.0
		PROTEIN, URINE	NEGATIVE NEGATIVE
		NITRITE, URINE	NEGATIVE NEGATIVE
		LEUKOCYTE ESTER	NEGATIVE NEGATIVE
		WBC/HPF	RARE 0-2 0-4 PER HPF
		RBC/HPF	RARE 0-2 0-2 PER HPF
		BACTERIA	<10 0 PER HPF
		CRYSTALS	PRESENT NONE SEEN
		CASTS	PRESENT NONE SEEN
		SQUAMOUS EPIS	0-2 0-2 PER LPF
		AMORPHOUS MATER	PRESENT NONE SEEN
		COARSE GRAN CAST	FEW 3-4 0 PER LPF
		FINE GRAN CAST	FEW 3-4 0 PER LPF
		HYALINE CAST	FEW 1-2 0-2 PER LPF
Student	02/12/2007-19:09:51	Order Basic Metabolic Panel	
		Na	142 135-144
		K	4.4 3.5-5.1

Figure 7. Patient Chart – Tests Tab.

By clicking on one of the icons along the top left, the student can choose other characters to talk to. In this scenario there are four agents other the patient. Each of them exists to provide opportunities to enact or explore important issues in the scenario. The ID Chief exists to highlight the importance of seeking expert consultation once a certain amount of evidence accumulates. The Tutor is prepared to note such consultation attempts. We have designed the interaction so that the important thing is that consultation be *sought*; however it turns out that the ID Chief is not reachable, so there is no *deus ex machina* to hand down an authoritative answer to the student. The Pharmacist exists to discuss appropriate treatment regimens for anthrax and to provide information about drug availability and the Strategic National Stockpile. The Hospital Administrator exists to support a conversation about systemic response options, in particular notification of other interested parties once an anthrax diagnosis is arrived at. Finally, the other ED (at Memorial Hospital) exists to provide a chance to highlight the importance on following up on information indicating that other people known to the patient may be suffering from a similar condition. Figure 8 shows two Player screen fragments focused on the simulation side capturing a couple of more snapshots of conversation, this time with the staff of the Memorial Hospital ED.

As the student plays the scenario in the left half of the screen, the Tutor is observing and offering proactive prompts, reactive feedback, and optional hints/explanatory comments in the right half. Figure 9 shows two Player screen fragments, this time focused on the Tutor side. In the first one, the student has clicked the 'Hint' button and the Tutor has suggested that getting chest imagery might be a good idea; note that the 'What' button is primed to give a more directive hint if necessary. In the second fragment, the student has acted on the Tutor's advice, and the Tutor has given positive feedback in response. In addition, the active hint has changed and the 'What' button is disabled again, until the student asks for a next 'Hint.'

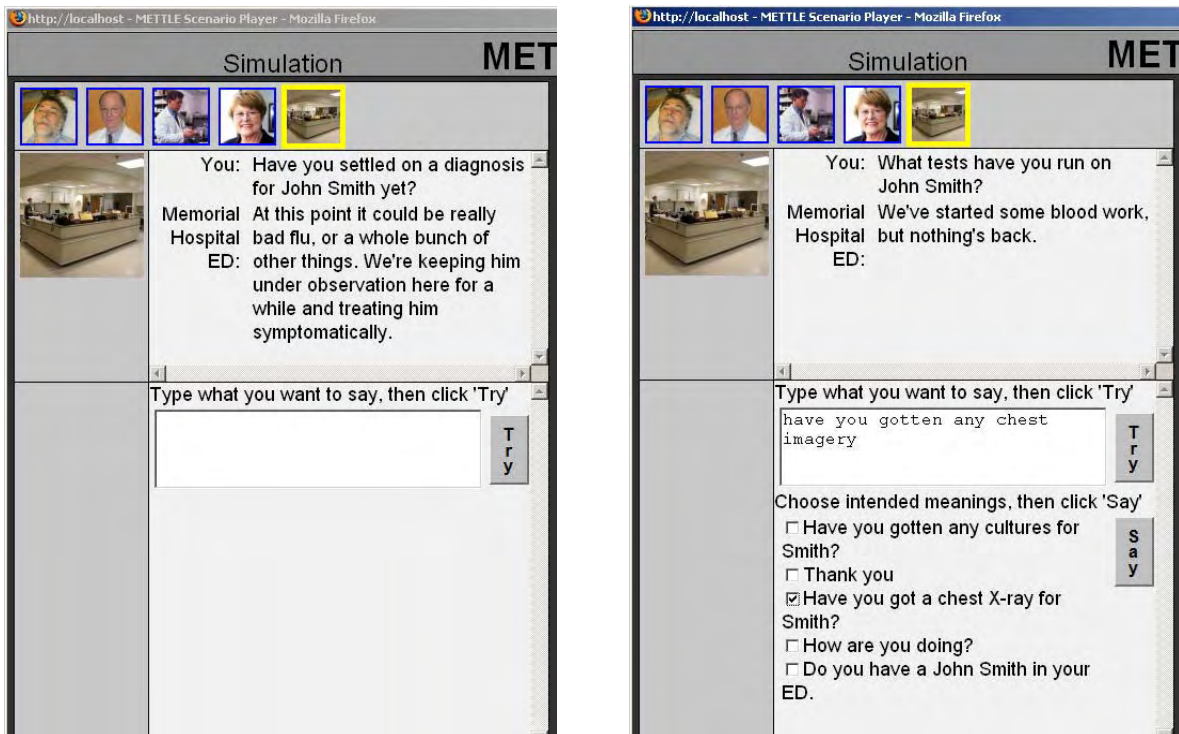


Figure 8. Interaction with Another Character.

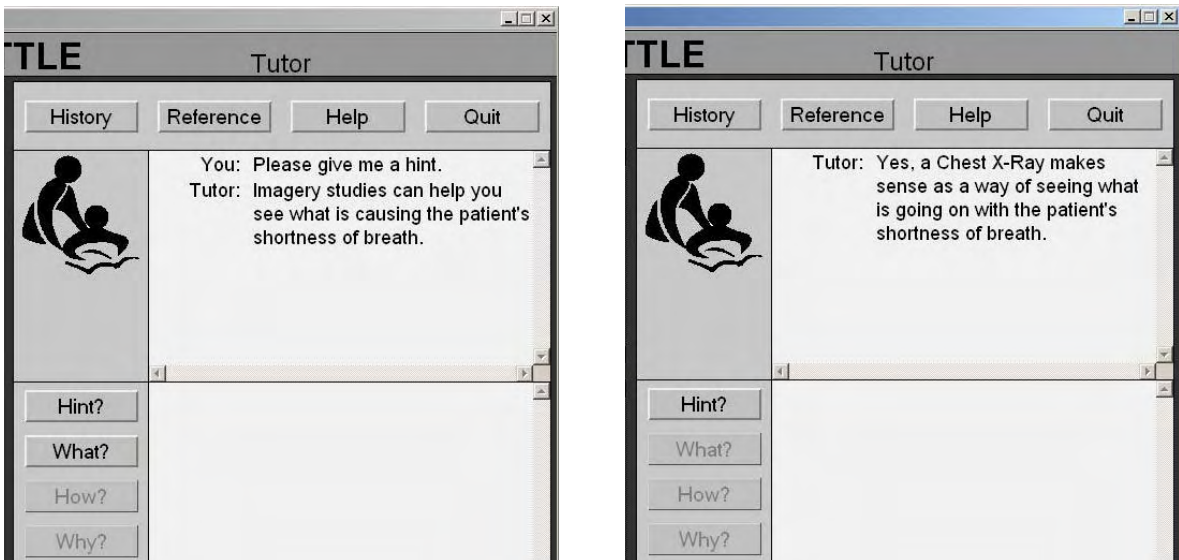


Figure 9. Tutor Hints and Feedback.

At intervals, the Tutor may go beyond simple commenting and hinting, and initiate larger scale dialogs that explore the student's diagnostic rationale. Figure 10 shows two more Player screen fragments from the Tutor side, showing two rounds of probing to elicit elements of the student's diagnostic differential in sufficient detail. These screens come from a relatively early stage of the scenario when many conditions (including many CBR Bacterial Agents) may reasonably be under consideration. At this stage, the Tutor may simply comment on the pros and cons of the items the student has chosen. At a later stage, when there is enough evidence for a diagnosis, the Tutor may drill down more deeply on the specific findings that make an anthrax diagnosis appropriate as shown in Figure 11.

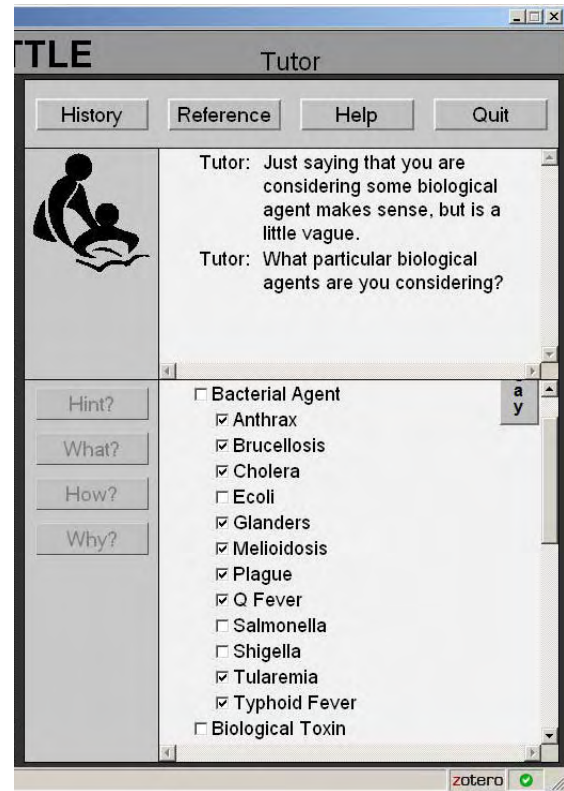
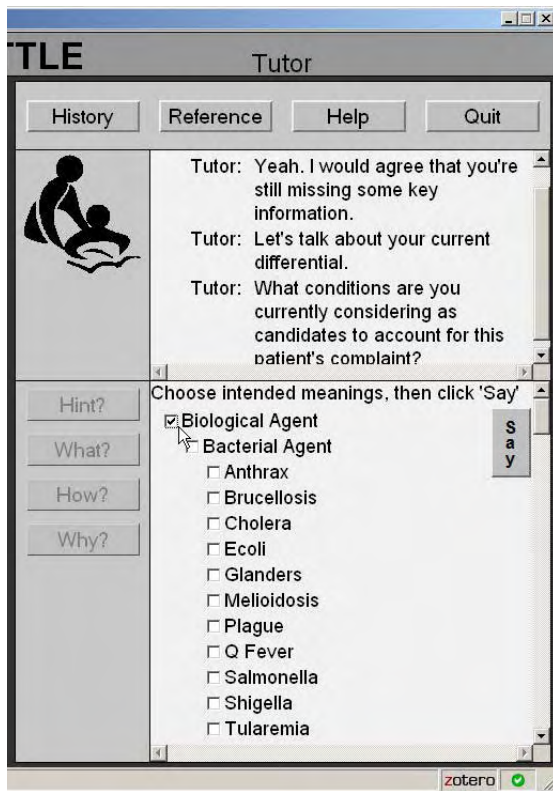


Figure 10. Tutor Elicits and Refines Student's Current Differential Set.

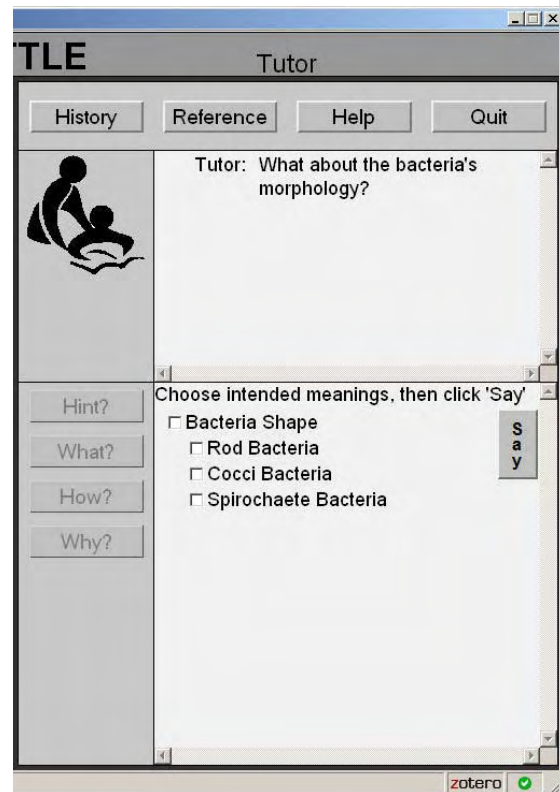
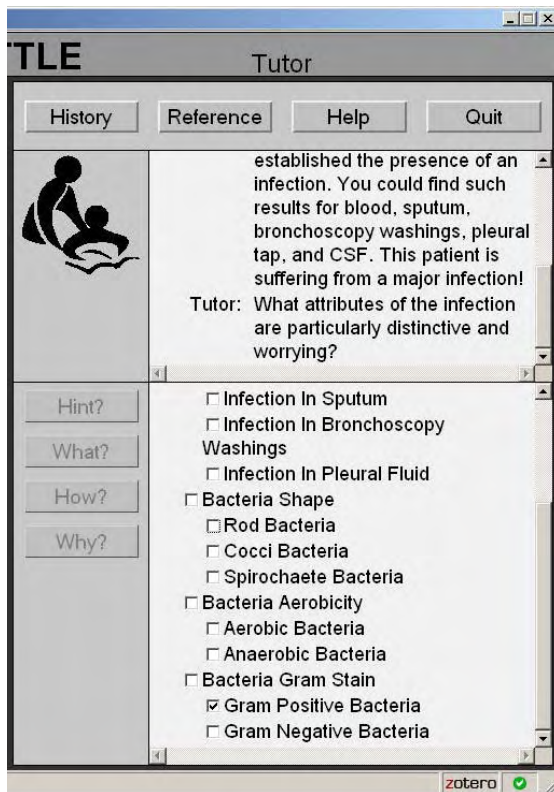


Figure 11. Tutor Queries to Highlight Critical Findings for Anthrax Diagnosis.

2.2.5 Authoring Capabilities

As described in our interim reports, we explored several different approaches to structuring and authoring scenario content during the course of the project. Structures explored included: (1) XML files (inherited from the Phase I prototype), (2) database tables (during the first development cycle), (3) KB objects, (4) special-purpose application files (both during the second development cycle), (5) human-readable behavior specification files, and (6) standard HTML/multimedia files (during the third and final development cycle). Authoring tools explored included (a) COTS desktop database forms, (b) SimVentive objects, events, behavior graphs, and GUI composition, (c) customized KB structure editing, (d) spreadsheet based authoring for high-volume special cases, (e) web-based forms, and (f) text-editing. Our final version of the system adopted structures 5 and 6, and tools d, e, and f.

One of the nicest features of our final approach is that it makes it possible to use a variety of tools that can be chosen to match (i) the demands of specifying some particular kind of content, and (ii) the preferences and capabilities of the person doing a given piece of authoring. This flexibility is facilitated by ensuring that each form of content has some human-readable form, and then providing translators to (and from) that form as needed, to present alternate faces through alternate tools. The details of the final authoring tools are described in Appendix D. Sample behaviors are described in Appendix E. Excerpts from extended dialog scripts are presented in Appendix F.

2.2.6 Authoring Methodology

In many ways, what is more important than the details of particular authoring support tools is the overall methodology for ITS authoring developed in concert with METTLE's scenario structure. In this section we review that methodology, and lay the groundwork for later estimates of the cost side of our cost/benefit assessment of METTLE. We describe a 10-step authoring process—a rational reconstruction of the substantially more experimental processes used for METTLE content development during this project. The first two steps represent *course-specific* up-front work to be amortized over successive scenario development efforts (with some recurring component, expected to become successively smaller). The next two steps represent *scenario-specific* design efforts. Most of the rest represent *scenario-and-actor-specific* behavior and media preparation efforts. The final step is scenario testing and refinement.

As will be suggested from time-to-time in the following discussion, we have identified four broadly different classes of work and expertise across these steps: (1) subject matter expertise, (2) instructional and scenario design and writing, (3) media production and general computer skills, and (4) facility with METTLE's behavior specifications. Obviously this does not necessarily require four different people on a production team: a computer-savvy SME with case-based teaching experience might easily cover all the bases (e.g. a professor at a medical school who is comfortable with computers). However, for most efficient use of an SME's time (which is typically the most limited and expensive kind of labor), it will probably make sense to distribute tasks across a team composed of people with different skills (and billing at different rates).

2.2.6.1 Course Design: Roles, Curriculum, Scenarios, Concepts, and Reference Materials

All instructional design should begin with a statement of who is to be trained and what they are supposed to be learning. Lacking solid contacts with a military organization offering an existing CBR medicine course and access to supporting subject matter experts, we chose to focus on civilian training needs, capitalizing on our good relations with an experienced Emergency Department (ED) chief who also serves as adjunct faculty at Harvard Medical School. Given our lead SME's primary expertise in the tasks of an ED physician, and in supervising and training other ED physicians, we put our primary focus on building training for junior ED physicians in dealing with CBR contingencies. Given our focus on physicians in a hospital setting, we identified (1) diagnosis, (2) treatment, and (3) systemic response as the core pieces of the curriculum (as opposed, say, to triage, first-aid, or incident command). Further we identified various categories of hospital staff as the key role-players to be trained and/or simulated—

primarily representatives of other hospital departments, such as Infectious Diseases (ID), Pharmacy, and Administration.

In developing curriculum, we initially worked from the Defense Medical Readiness Training Institute's (DMRTI) October 2003 "Chemical, Biological, Radiological, Nuclear, and (High Yield) Explosives (CBRNE) Training – Standards of Proficiency and Metrics" (DMRTI, 2003) document. We entered a large subset of those standards in METTLE for use as curriculum points. As we developed our sample scenario in detail, however, we found that the DMRTI standards were too broad and somewhat too fact oriented for METTLE's purposes. The breadth made it difficult to find the right part of the taxonomy when seeking to attach curriculum points to tutor annotations. The knowledge-centric nature of many of the standards meant that too many tutor annotations would be left addressing generic items such as "List the clinical signs and symptoms associated with each agent" (DMRTI's ELO-8.41-1). We preferred to develop more specific performance-oriented items such as "Demonstrate recognition of anthrax" as well as even more explicitly control-oriented items such as "Blood samples before antibiotics" or "CT before lumbar puncture." Within the bounds of the identified area of performance (e.g. diagnosis, treatment, and response to CBR contingencies) the specific set of curriculum nodes is expected to evolve based on SME identification of key points of proper performance during elaboration of a set of scenarios.

As characterized in earlier reports, we initially identified a scenario space intended to cover the universe of DoD-identified CBR agents, with scenario variants designed for play by (and instruction of) different hospital medical team roles (e.g. ED-physician, ED-Chief, ED-Nurse, ED-Orderly, ID-physician, Radiologist, Cardiologist, Pharmacist, Lab-Tech, Administrator). This implicitly suggests a scenario corpus of up to 1000 scenarios (roughly 100 agents by 10 roles). The reality, however, is that not all of these specialties will play an equally significant role in dealing with CBR contingencies, and given limited resources, it seems most critical to invest in physician training.¹ Likewise, though there may be on the order of 100 CBR agents of concern, many of them fall into families, differences among whose members may not be significant to the target trainees (e.g. 11 different radiological isotopes); in many cases a paramount goal is to know enough to recognize that something unusual is going on, and to figure out enough about what it might be to identify appropriate specialists with which to consult. CBR agent and role are not the only possible dimensions for defining scenarios; scenario complexity can be graded in other ways, such as (a) number of patients, (b) mix of diseases, (c) stage and/or severity of disease, (d) typicality or atypicality of symptoms and ensuing findings, (e) other complicating factors. The number of scenarios that can actually be developed will depend not only on the resources available, but on the costs required to build a scenario; estimating those costs is one of the goals of this section. The number of scenarios that any given student might actually use is another question we will take up later.

To an even greater extent than with the curriculum, the conceptual taxonomies that prove useful in exercising and tutoring the skills covered by the curriculum must be worked out incrementally along with the scenarios. The general categories of concepts (or as they are called in METTLE, the concept *spaces*) become clear quickly enough. As shown in Figure 56, the spaces represented in the tabs of the Concept Editor include Conditions, Findings, Tests, and Treatments, along with patient Management, and system Response options. These concept spaces obviously align with the stated training goals of the system.² Some key concepts can be mined from authoritative documents in the field. For instance, our taxonomy of CBR conditions was built based on the U.S. Army Handbooks of Biological (USAMRIID, 2005),

¹ We note that our project focused on ED physicians, but feedback suggests that the training should actually be targeted at physicians more broadly, including in particular primary care physicians, who like ED physicians are on the front lines of general patient care, and likely to encounter early CBR casualties.

² Of course the same software infrastructure can be applied to teach other medical and professional topics, requiring different curriculum and concept spaces. An example would be current work for the U.S. Army on interagency collaboration for Stability, Security, Transition and Reconstruction (SSTR) operations, where historic operations, organizations, goals, and tasks so far appear to be the central concepts.

Chemical, (USAMRICD, 2000) and Radiological (MMOAFRRI, 2003) Casualties, supplemented by other Field Manuals and Texts; our set of patient interview questions was largely drawn from DeGowin's *Diagnostic Examination* (LeBlond, DeGowin, & Brown, 2004); and our basic space of diagnostic tests was built from a representative hospital order sheet, while core treatment options were drawn from drug formularies. On the other hand, our space of non-CBR conditions must be elaborated based on a per-scenario research to determine conditions that might plausibly be included in a diagnostic differential for any given presentation of any given CBR agent.

Finally, in our move to a fully web-based system, we recognized the opportunity to support professional learning by, wherever possible, linking from key concepts to authoritative on-line information sources on those concepts (e.g. diseases, treatments, etc.). This is an area where much additional useful work can be done, by integrating web search and text extraction technologies with the ITS in order to more broadly and efficiently provide links to up-to-date reference materials that relate to scenario-highlighted challenges.³

2.2.6.2 Conventional Behaviors, Forms, and Related Interactors

For METTLE, with its primary focus on CBR diagnosis, we have so far evolved a space of about 300 patient interview questions, 112 diagnostic tests, 64 examination actions, and 57 treatment actions. We expect these numbers to expand as additional scenarios are built, but probably not precipitously. The system has a taxonomy of approximately 100 CBR agents, supplemented by about 20 conventional medical conditions. The universe of conventional conditions can certainly expand over time, as new ones need to be introduced to flesh out a plausible differential set for each CBR agent under consideration.

Generally speaking, for each of these collections, in addition to simply identifying their elements and organizing them in terms of taxonomic and other relations, some additional preparatory work is required to set up the METTLE environment for their effective use. This work is of three types:

1. In essentially all cases, when new elements are identified, it will likely prove useful to expand the repertoire of default behaviors for some class of actor (in our experience, the focus has been on the patient character). Thus, for instance, recognition that we missed a common patient interview question would require creation of a new default script line that specifies a range of cues that will let a simulated patient recognize when a student has attempted to ask that question. Similarly, incorporation of a new test or treatment will require generation of a default script line to recognize the ordering of same.
2. In cases where new concepts are introduced that fit into a concept-space that appears on order forms, it will also be necessary to modify the related form to include the new option. While in theory such forms could be auto-generated from the underlying concept space, we chose to allow (require) authoring intervention in order to allow finer-grain control over the look of the forms. In Appendix C, Figure 23, Figure 24, Figure 27, and Figure 36 show four different order forms that illustrate varying amounts of semi-custom layout (e.g. control of ordering, grouping, and side-by-side layout).
3. In the least frequent case (e.g. when shifting into new areas of application) it might be necessary to add a new form, or perhaps even invent a new kind of interactor. In the current state of the system, any such change calls for programming skills, either to expand the pop-up menu system that offers form choices, or to add a new pop-up window to display a class of forms in, or to define entirely new interactive behavior. With some additional development work, mere changes to menu structures and form inventories should not require such intervention.

³ This is precisely one of the major areas our current SSTR operations training work addresses, focusing specifically on making better, more timely use of SSTR "Lessons Learned" from the field.

As with step 1 above, the bulk of this effort should be concentrated during the up-front course design period, and during initial scenario development. The amount of time spent making such modifications on a per-scenario basis should drop with each scenario.

2.2.6.3 High-Level Scenario Design: Situation, Patient(s), Points, Agents, Scenes

Now we turn to the design of a particular scenario. First we need to determine what part of the overall scenario space sketched out during course design is the next most important area to cover: e.g. what CBR agent, what level of difficulty, what general themes, what core curriculum points. This is the stage at which, for instance, we determined that we were going to develop a scenario involving a single (index case) of anthrax with primary curricular emphasis on detection and diagnosis skills, and secondary coverage of treatment and response. It is the stage at which we decided on the exposure back-story, how long the disease had been incubating, how sick the patient would be, what might be happening to other exposed individuals, and what environmental or personal factors might intervene to complicate diagnosis. This is a good time to prepare a draft of any briefings, including initially given information about the patient (i.e. what would be included on the patient's chart at the start of the scenario).

This is also the stage to identify the cast of other simulated actors who will be available to the student during the scenario. Actors should generally be introduced to enable coverage of some relevant part of the curriculum. For example, another patient (or another ED) can be introduced to test whether the student follows up reports of other people being sick; an Infectious Diseases (ID) specialist can be introduced to provide an opportunity to test the student's decision-making about when to seek expert consultation; a pharmacist can be introduced to support choices of treatment, and to learn about longer term systemic responses to support prophylactic treatment on varying scales.

Finally, this is the stage at which decisions can be made about the overall scope, shape, and degree of dynamism required for the scenario. In many cases, it will suffice to build a scenario around what is essentially a single snapshot view of a patient (maybe supplemented with some record of earlier historic states); this is a common approach in medical case presentation. Sometimes, however, it is interesting to present multiple snapshots because evolution of the disease process is revealing, there are different lessons to be learned at different stages of the disease, or earlier interventions can produce different sequelae with interestingly different problems. In the first situation a scenario with a single scene will suffice, in the later situations multiple scenes may be useful, perhaps even with a conditional branching structure (based on student actions in earlier scenes).

When deciding how many points to pack into a scenario over how many scenes, it is important to keep in mind the implied costs—both for developers and for students. For developers, the costs of behavior scripting and associated media preparation (see below) grow with the number of different health states a patient can be in (requiring different answers to questions, or different test results), and the costs of tutor scripting grow with the number of significantly different knowledge states the student can be in. A two-scene scenario might be twice as expensive to develop as a one-scene scenario (and might take twice as long to play, ideally while covering twice as many curriculum points). For students, the costs of a scenario are measure in the time spent playing. Subjects in our final evaluation study typically spent between 45 minutes and an hour playing our single-scene anthrax scenario. In the military operations world, where there is substantial experience with simulation-based training, two hours is often considered the limit on how long a single exercise session ought to last (in the interests of maintaining student focus). Recommendations gathered during our formative evaluation suggested that in the medical world it might be worth thinking in terms of a two-hour Continuing Medical Education (CME) unit that would be constructed out of four or five much shorter (20-30 minute) scenarios.

Taking a nominal one-patient/one-scene/one-hour scenario as a kind of benchmark, we believe that the high level scenario design ought to consume about two person-days, ideally involving brainstorming and debate between an SME and an instructional designer. Preparatory for this design work, the SME ought to either have solid command of the literature on the underlying condition, or spend a day

researching and preparing. For example, in the case of anthrax there is a substantial literature, including a useful summary of actual case histories (Jernigan, et al., 2001). A key issue that requires SME judgment and research is to identify the set of alternate conditions that could reasonably be (or have historically been) confused with the condition in question. In the case of anthrax, we actually found a paper that reviewed the differential generated by a sample of doctors when confronted with anthrax symptomology (Temte & Zinkel, 2004).

2.2.6.4 Detailed Scenario Design: States, Evidence, Flows

The next set of decisions to be made about scenario design fill in more details within the framework provided by the high-level design.⁴ The key issue is to define the interestingly different flows through each scene. The primary mechanism for flagging changes within a scene is assignment of *states* to actors. There are two main uses for these state flags: (1) to indicate changes in an actor (e.g. if we want a patient's physical condition to deteriorate during the course of a single scene, or a simulated colleague's emotional reactions to change based on earlier interactions), and (2) to indicate changes in tutor expectations about what the student should know and be thinking regarding another actor (e.g. if we want the tutor to offer different comments and assessments depending on diagnostic actions the student has taken for a particular patient during the scene).

The second use is obviously more interesting from the perspective of automated assessment and instruction—the heart of an ITS. We suggest that the author look for a small set of key diagnostic findings that ought to affect a student's thinking. In the case of anthrax, such findings would include (1) evidence of gram positive bacterial infection, (2) evidence of gram positive bacterial infection at sterile sites, (3) imagery evidence of pneumonia and other more specific pulmonary abnormalities. Assigning patient states on the basis of such findings—as a way of reflecting tutor expectations about the student's knowledge of the patient's condition—is a convenient way to centralize a set of tests for combinations of diagnostic actions that will control a range of tutor behaviors;⁵ clearly knowing, for instance, that the student has sufficient evidence in hand to arrive at a diagnosis should affect the tutor's behavior.

The idea, then, is to design a small graph of significantly different knowledge states a student might be in regarding a patient's condition. Defining those states requires identifying the evidence a student might gather that would give them that knowledge, and the actions they might take to get that evidence. For instance, there are two different tests that can reveal a gram positive infection, which is suggestive but inconclusive because they sample from non-sterile sites (gram stain of sputum, and gram stain of bronchoscopy washings). There are five tests that can reveal a much more informative gram positive infection at a sterile site (blood culture, Cerebrospinal Fluid (CSF) culture, blood gram stain, CSF gram stain, and pleural tap gram stain). Finally there are two imagery studies that can reveal important pathologies of the pulmonary system (chest X-Ray and chest CT).

One place where tutor sensitivity to meaningful constellations of gathered evidence is particularly important is in Socratic dialogs that probe student's evolving understanding of the situation. METTLE provides machinery to approximate the kind of dialog a medical instructor might have with a student: eliciting a current differential, critiquing the student's hypotheses, and suggesting (or eliciting) ideas for gathering more evidence that could clarify the picture. Such differential diagnosis dialogs must fit the key evidence that has been uncovered so far. A related use for tracking student knowledge states in this way is to control the overall flow of the scenario: once a student has uncovered all the relevant diagnostic

⁴ Note that though the discussion of this 10-step methodology reads like a classic “waterfall” development method, it is not really intended to be executed in that way. In general, discoveries made during later steps may cause authors to go back and revisit decisions made earlier on—to extend curricular or conceptual hierarchies, adjust default behavior sets, refine or elaborate the scenario structure, revise scenario behaviors and tutoring, and so on.

⁵ The reasons we propose using *patient* state markers (rather than, say *tutor* state markers) are that (a) the tutor is forming judgments about the student's state of knowledge about a patient, and (b) most tutor behavior is actually specified as annotations on other actors' base behaviors.

information (and especially once they have been through a Socratic dialog that helps them understand the import of that evidence), the diagnostic portion of the scenario is effectively over.

Of course not all aspects of scenario control and flow need be based on such state changes. Tutor (or other agent) behaviors sensitive to less complex patterns of student activity or less pervasive in their effects can be encoded as simple tests assigned to specific behaviors. For instance, one of the learning points built into the anthrax scenario has to do with following up evidence of correlated illnesses across patients who share some connection. The tutor behaviors that prompt the student to pursue this direction are built into a pair of simpler behaviors: one that prompts the student to ask if the patient knows anyone else who is sick like them, and a second that (assuming the student has asked that question) prompts the student to follow up that lead by calling the hospital where our patient's cousin is known to have gone.

Again for our nominal one-patient/one-scene/one-hour scenario, we expect that this detailed design could consume another two-person days of effort by SME and instructional designer. In all, the design phase of scenario development would be expected to take about a person-week: for the SME, one day of research, one day of high level design, and one day of detailed design; for the instructional designer, a day each of high-level and detailed design.

2.2.6.5 Basic Scenario Behaviors

Having identified the scenario patients, their conditions, key learning objectives, cast of characters, scene structure, important states, and major aspects of scenario flow, we are now in position to start generating specific agent behaviors. The patient is likely to have the most behaviors, but the bulk of those are partially pre-specified (e.g. their cues are defined in the default behavior files). In our anthrax scenario the only patient behaviors created totally de-novo were those dealing with questions about the specific acquaintances who were also sick, and those dealing with scenario flow control; in all, it came to about 20 totally novel behaviors. This is in contrast to the roughly 550 conventional behaviors built on patient defaults.

As noted earlier, the system currently has a library of about 300 conventional diagnostic interview questions. Authoring the customized responses to those questions requires some mild creative writing combined with a solid medical understanding of how a patient with the target condition would likely respond. For any target condition, the vast majority of the questions are either irrelevant or present opportunities to introduce pedagogically useful distractions. As one example, in early scenario design discussions we considered the possibility of having our patient be on drugs for mild schizophrenia, simply because there is a syndrome associated with such drugs that presents with some similarities to anthrax.

An initial spreadsheet can be prepared with all of the default patient behaviors pre-entered. For a first pass, the author need only run down the list of questions and enter the text of the particular simulated patient's responses. Likewise the author can run down the list of examination actions and enter the significant findings that should be reported to the student and logged into the patient chart. Some other classes of pre-defined patient behaviors (in particular, tests) are more media-heavy and are discussed in the next sub-section. Totally scenario-specific behaviors requires a bit more effort to specify, since the cues have to be entered as well as the responses; still custom interview question cues are not much harder to create than the custom responses. As an example, the custom behavior watching for a query about our anthrax patient's last contact with their cousin was assigned the cue:

```
(or (hear "When did you last see your cousin?")
    (hear "How long ago were you last with your cousin?")
    (hear "How many days has it been since you spent
        time with your cousin?")
    )
```

For other agents, at this point, all of the behaviors are custom, beyond a small collection of standard social conversational scaffolding (e.g. “hello,” “how are you doing,” “thank you,” and “goodbye”). This should change over time, as we develop a better sense of the recurring conversational patterns associated with other roles. For instance, there are probably a very constrained class of questions one would want to ask a hospital pharmacist, relating to likely scenario goals such as (1) identifying appropriate drugs for particular conditions, (2) checking stocks of those drugs and any reasonable alternatives, and (3) exploring sources of and conditions on larger supplies. We note that doing a good job with such interactions would benefit from extensions to the `hear` and `say` Interpreter sublanguages to (a) allow inclusion in input of references to concept spaces whose concepts have been assigned recognition terms (e.g. common names of drugs in a formulary), and (b) allow substitution on output of values looked up in tables (e.g. stocks kept on hand, volumes available, delay times for future shipments, etc.).

Disregarding media preparation (discussed below) we believe it should take on the order of two to three person days of spreadsheet work and review to prepare the basic behaviors for a patient. Each additional character could take an extra half day to day depending on the complexity of their interaction (e.g. how many different curriculum points are potentially addressed). In round numbers, we budget basic scenario behavior development at a person week, requiring a mix of SME and instructional design skills (with simple METTLE behavior authoring).

2.2.6.6 Scenario Media: Briefings, Images, Test Results, Sounds

The scenario briefing file is a standard HTML file whose content is determined by the scenario design. It should cover the basic who, what, where, when sorts of questions to orient the student to their role and situation in the simulation. It can seed some clues, distractions, and/or complications as seems appropriate to the scenario designer. In our Anthrax scenario, the briefing set the scenario in a northeastern city during a flu outbreak (distraction), and mentioned that the patient had a sick cousin he’d been out with recently (clue). When providing introductory material about a patient, it should generally match the information seeded on the patient’s chart. Individual scene briefings are optional, and were not used for the single scene of our Anthrax scenario.

A set of about 70 patient images to support the physical examination can be shot in an hour (of two people’s time) if they do not require any special moulage make-up, and might require an additional two hours to download, review, select, resize, and rename. Creating the overview image (composite front and back full body views) and defining the hot spots with appropriate names to map to the detail image files and the pre-defined examination actions might take another two hours. We used two free open source products for these purposes: the GIMP image processing program for resizing and composing images, and the Inkscape SVG editor for defining hot-spot regions (using its XML view to assign desired names to rectangles drawn over the base image). Finding or shooting and then preparing three variants of each actor image could also take a few hours. In all, character image creation could take a day or so for a scenario if there is no significant re-use.

Generating test results presentations is substantially more complicated. First, an SME must determine which tests from the pre-defined universe of tests could plausibly be relevant to the case being developed. They must also consider if there are any unusual tests, not yet in the system’s repertoire that should be added (and if so, a bit more time is required to extend the systems testing concept space, and to fit the new test into a relevant order form). For our Anthrax scenario, we identified 38 tests (out of a universe of 112 tests drawn from a representative hospital’s in-house test order form that were known to the system). Having determined the relevant tests, the SME must then determine the format and content of reasonable result. The format can be copied from existing hospital reports. The content must be drawn from experience and the relevant literature. While most tests produce tabular results that can easily be authored as simple HTML tables, some—notably imagery studies (e.g. X-Rays and CTs) and strip recordings (e.g. EKGs)—produce graphical results, generally accompanied by expert interpretation in textual form. Finding appropriate images to incorporate into scenario test result displays requires access to a good

medical library, or willingness to Google and copy freely. In all, preparing the media for a simulated patient's possible test results can take a day just on the production side, formatting, testing, and proofing the various HTML files. The SME time can also easily amount to a day or more, depending especially on how much time they have spent in advance learning the relevant literature in support of scenario design.

To illustrate METTLE's ability to exploit sound, we chose to record spoken versions of all the simulated patient utterances for the Anthrax scenario. This came to nearly 300 .wav files. The same could have been done for all the simulated agents, especially if we had a stable of willing voice actors to draw on. Of course, if having audio was deemed important, but quality was less of an issue, automated speech generation could be used as well. Our longest patient utterance was about 10 seconds long, with most being substantially shorter. Allowing about a minute to record each utterance, the 300 patient utterances could be recorded in about five hours. If such recording is going to be done, it is worth leaving it until relatively late in production in order to minimize the amount of re-recording that becomes necessary as agent utterances are edited or tweaked in the course of scenario refinement.

In all, the various kinds of scenario media production discussed in this section can again be budgeted at another person week, requiring primarily media production skills, but also a day of SME time for defining test results.

One final issue worth keeping in mind during media production is management of media bulk. The sizes of image and sound files can make a substantial difference in network delivery times, especially over wide-area networks. We have found that saving JPG images at 85% quality is adequate, and substantially reduces image storage size in comparison to higher quality levels. Likewise, we have found that saving WAV files as PCM 22.050 kHz, 8 Bit, Mono results in reasonable speech quality with acceptable file sizes. It should also be possible embed short videos (e.g. AVI or WMV files) though both bandwidth and decoder compatibility issues become more severe for video.

2.2.6.7 Tutor Behaviors

A subset of all simulated actor behaviors bear directly on course curricular objectives and the critical flow of the scenario. Given the size of the pre-defined patient behavior set, the relevant subset in any given scenario is likely to be relatively small. As described earlier, but bulk of METTLE's tutor behaviors are specified as annotations on base actor behaviors. In this step, the behaviors prepared earlier should be scanned, and the ones critical either to learning or to maintaining progress through the scenario identified. Such behaviors will be annotated with curriculum points, tutor comments (proactive prompts, available hints and explanation, and reactive feedback), and relevance conditions.

While there may be very little in the way of strict procedures that a student should be following, there are generally conditions determining the relevance or importance of their taking particular actions. These conditions should be translated into observables in the METTLE environment (most frequently into checks on the scenario history to see whether or not the student has yet taken some actions) and turned into tutor tests (e.g. coded in Columns S-X of the authoring spreadsheet). When does it become relevant to order basic blood (cultures, gasses, and metabolism panel) and urine tests? Pretty much as soon as the patient is set up in the ED and an IV established. When should a test of CSF be ordered? For our anthrax scenario patient, basic test results should suggest the possibility of meningitis and provide motivation for such tests, but even then, CSF should not be collected until a head CT has been reviewed. Such tests can be written in terms of combinations (and, or, not) of line test operations.

Given behaviors worth discussing and the conditions under which those behaviors are appropriate, METTLE provides for seven different kinds of tutor utterance that fall into three categories:

- **Proactive Prompts:** When defined, a pro tutor utterance will be output at some specified time (when, Column R) after the relevance conditions for the behavior become true.
- **Hints/Explanations:** When defined, a cascade of hint, what, how, and why utterances (not all of which need be provided) will be offered to the student (by enabling the corresponding tutor

buttons on the Player page) after the relevance conditions for the behavior become true. A limited set of hints will be offered at any one time, with the tutor rotating among those associated with the highest ranked active behaviors (#, Column Q).

- **Reactive Feedback:** When defined, a `yup` or `nope` utterance will be output depending on whether, (for `yup`) the student triggers the associated behavior after its relevance test becomes true and optionally before the when time expires, or (for `nope`) the student triggers the behavior either before it is appropriate, or optionally after the when time expires.

From an authoring perspective, the questions are: (1) which kind of utterances to associate with which behaviors, (2) what delays/time-limits to specify, and (3) what ranks to assign. In general, we tend to use proactive prompts (often combined with hints/explanation on the same or related behaviors) to try to ensure that students don't get bogged down and fail to make progress because they miss out on some critical step in the scenario. We use hints/explanations relatively frequently, discussing many actions that a reasonable physician might take at various stages of the scenario, often explaining why they would be reasonable in the context (given the associated tutor test is true). We use reactive feedback fairly sparingly to highlight important steps or missteps; our assumption is that physicians do not want to receive too much congratulation or critique from a machine.

Assuming basic behaviors scripting takes on the order of a week, we expect that scripting the tutor utterances associated with selected behaviors should add about another week. While far fewer behaviors require such annotations, the preparation of tests and utterances are generally more complex and time consuming than the preparation of basic cues and responses. Where the behavior authoring requires knowledge of the disease processes being simulated, and the kinds of interaction desired with other characters, the tutor annotations require careful thinking about what student behaviors are desired (or undesired but likely), under what conditions, for what reasons, and bearing on what curriculum points.

2.2.6.8 Extended Agent-Initiative Discussions

There are two main categories of extended (agent-initiative) dialogs that are especially useful in the context of METTLE: (1) discussions driven by the tutor to explore diagnostic differentials, and (2) discussions driven by other simulated authority figures to explore other aspects of student performance and/or reasoning. The unifying idea is that we expect such dialogs to be driven by a simulated agent, and that generally means they are “in charge” by virtue of superior knowledge or status in some domain of discourse. Possible exceptions here might include extended dialogs with concerned family members, or media representatives, who would be neither authorities nor experts, but could reasonably take the initiative and drive an “information seeking” dialog that requires the student to explain some aspect of the situation; such dialogs seem less useful, either in terms of fit to real-world tasks, or in terms of the agents ability to critique or correct student mistakes during the dialog.

As suggested earlier, we believe it is worth preparing a diagnosis exploration dialog for each major state of student knowledge, plotted out in terms of state transitions during detailed scenario design. Such dialogs can have a relatively standardized structure: (1) elicit the student's current differential (probing repeatedly as needed to get them to think more broadly or specifically), (2) review the likelihood of items that they included given the current state of evidence (especially noting conditions that should be considered low probability or already ruled out), (3) review items they did *not* include that should be in the running (especially noting the evidence that supports them), and (4) along the way, highlight additional evidence that could be collected to bolster or undermine currently plausible conditions. One such dialog can be put together in a few hours; by virtue of content overlap and reuse, a related set of four or five such dialogs over a reasonable sized differential can be put together in a day. For the anthrax scenario, we considered a potential differential space that included all CBR agents (though of course many whole classes could be ruled out quite quickly) as well as two dozen more conventional conditions. Elaborating these dialogs—making them more Socratic and less assertion-based when it comes to the details of dialog parts 2-4 called out here—can take an arbitrary amount of additional time. It may be

worth investing another day or two to strategically elaborate pieces of these dialogs to *push the students to flesh out arguments* for the relevance or irrelevance of particular conditions given available evidence, or to *elicit suggestions from them* for additional evidence to be gathered.

In addition, there may also be characters whose reason for inclusion in the scenario is to drive exploration of a set of issues through discussions, perhaps because either (a) talking about some class of action (planning or advising) is more realistic for a given student role than actually carrying them out, or (b) discussing actions provides greater opportunity to get at underlying rationale (and hence more scope for training decision-making) than does simply performing actions. In the anthrax scenario, the hospital administrator has a short dialog that plays this role with respect to decision-making about notification of other organizations, the media, and the public. While ED physicians are not typically responsible for emergency communications strategies, having them understand the rationale for such strategies (including why they should be circumspect about what they say and to whom) is a reasonable training goal.

Finally, we recommend that the tutor be given a final scenario wrap-up dialog that serves as a kind of summary and after-action review. This can be as context-sensitive and/or interactive as the authors care to make it. However, for simplicity and economy, in our anthrax scenario we built this final dialog in a more conventional straight lecture format.

In all, we recommend budgeting a week to produce a set of extended discussion scripts. This is the most technically demanding aspect of METTLE authoring, and the current tool set does not offer as much support as we would like; we use COTS text editors to create these scripts (see the discussion and samples in Appendix F). Fortunately the Discuss layer uses essentially the same Interpreter language as does the Enact layer, so the learning curve is substantially reduced.

2.2.6.9 Control Behaviors

The final set of behaviors needed to stitch a scenario together and ensure appropriate flow are what we call control behaviors. These are relatively few in number, but may require more thinking like a programmer than other types of behaviors. We have identified some common (non-exclusive) categories of such behaviors:

- **Behaviors that change (and test) agent states:** These behaviors are the implementation of the agent (often patient) state-based control scheme we advocated in 2.2.6.4. One set of behaviors is needed to watch for the constellations of conditions that cue state-changes and then to effect those changes. The state change itself automatically affects which scripts are active for that actor. However, if the state represents a state of *student knowledge*, then other actors' behaviors may need to be modulated as well, and doing so will require explicit control behaviors in those other actors' scripts.
- **Behaviors that prompt essential actions or launch important dialogs:** These behaviors serve to advance the overall flow of the scenario and ensure that pedagogically central activities either do not get skipped, or happen at a reasonable time. Examples include behaviors that launch the diagnostic differential discussions appropriate to a given state of student knowledge at the point where the student does anything that might be construed as indicating they have reached some kind of conclusion, and behaviors that launch the tutor's final wrap-up discussion.
- **Behaviors that disable entire (contextually inappropriate) areas of action:** There are phases in a scenario (especially tied to states of student knowledge) during which certain classes of action may be irrelevant. Behaviors sensitive to those states can be used to effectively disable otherwise available aspects of scenario functionality. Examples include preventing interaction with some simulated actors before sufficient information has been gathered to make those interactions meaningful, or critiquing the execution of various management or response actions before enough information is in hand.

Given a solid design for the scenario and its flow, the creation of these control behaviors need not be a major task. We would expect it to take about a day, though we recommend this work specifically be handled by someone comfortable with METTLE and a programmer's mindset. In this context, we also revisit the first two tasks, which were described as being primarily tied to the overall course, rather than to any one scenario. However, we noted a number of aspects of those tasks that were likely to require some elaboration with each new scenario (e.g. extending curricula, concept spaces, forms, and default behaviors), and which would benefit from comfort with METTLE technology. Accordingly, we suggest budgeting for a week of time to cover this combination of tasks.

2.2.6.10 Testing and Refinement

We assume that a reasonable testing regimen would include on the order of 8 to 10 sample students each spending 1-2 hours playing and then discussing the scenario (a total of about 2 days of student time, requiring about 3 days observer/analyst time). We add another week for authoring tweaks and fixes based on what is learned from such trials.

2.2.6.11 Summary of Authoring Costs

Table 1 collects our level-of-effort estimates for each of the tasks from the previous sub-sections, and also suggests rough allocations of effort across the different categories of development labor called out in the introduction to 2.2.6. In broad terms we believe it will take around 8 person-weeks to prepare a nominal one-patient/one-scene/one-hour scenario. The costs implied by these labor estimates vary considerably depending on whether you assume commercial, academic, or government labor rates.

Table 1. Major Scenario Development Tasks with Estimated Level of Effort by Labor Category

Task	Level of Effort	SME	Instruct. Designer	Media Producer	METTLE Tech	Sample Students
Design	1 Week	50%	50%			
Basic Behavior Authoring	1 Week	20%	80%			
Media Creation	1 Week	30%	10%	60%		
Behavior Annotations	1 Week	20%	60%		20%	
Extended Discussion	1 Week	20%	20%		60%	
Control Behaviors & Course-Wide Tech Content	1 Week		20%		80%	
Testing/Refining Scenario	2 Weeks	20%	20%	20%	20%	20%
Totals	8 Weeks	9 Days	14 Days	5 Days	10 Days	2 Days
\$ (commercial)	\$45K	\$18K	\$14K	\$2K	\$10K	\$1K
\$ (academic)	\$30K	\$12K	\$10K	\$1K	\$7K	\$0K

We are comfortable with these estimates, understanding that they build in assumptions such as (1) a new scenario is largely compatible (in content and structure) to the existing course and scenarios, (2) the staff are experienced in METTLE scenario development, (3) no time is spent trying to improve tooling or processes.

2.2.7 Final System Evaluation

In this section we report on the results of our METTLE Final Evaluation study, which provides very preliminary data on the possible benefits of a deployed METTLE system. As of 5/13/2008 we had received responses from 11 physicians, including one who was unable to get the software to work on their Windows machine.

2.2.7.1 Evaluation Method

We identified a random selection of emergency physicians from on-line sites, and mailed out letters soliciting their participation in our METTLE Evaluation Study. The letter explained the nature of the

study, the amount of time expected (up to 2 hours), and offered \$100 compensation for their time if they participated. In response to 2500 such letters⁶ we ultimately received indications of interest from 55 physicians⁷ in the form of emails to a special address we had set up for the purpose.

With a goal of 20 respondents, we mailed out 30 evaluation packets. Each packet contained (a) a cover letter explaining the details of the evaluation process, (b) a CD ROM containing the METTLE Server software with supporting video introductions to its use and related browser configuration issues, (c) an Evaluation Feedback Form, (d) a Payment Sheet, and (e) a stamped return envelope. The Evaluation Feedback Form is included as Appendix H of this report.

The materials in the packet also included Stottler Henke email and telephone contact information for use in the event the physicians ran into difficulty with the software. We ended up exchanging support email with 5 physicians, and speaking with 2 of those 5. To date we have identified 4 issues that led to technical difficulties, the first 3 of which we were aware of and had tried to account for in the instructions we distributed: (1) the METTLE Server needs to be copied from the CD onto the user's hard drive so that it can write intermediate files as it runs, (2) the software is only compatible with relatively modern browsers including Internet Explorer v7 and Firefox v2, (3) the browser must have "pop-up blocking" turned off for the software to work, and (4) one user reported incompatibility with Windows Vista Ultimate edition based on a conflict or restriction related to use of the standard web server port 80. To address issues 1 and 4, when we sent out our second wave of evaluation packets, we offered subjects the option of *not* installing the METTLE Server, but rather running off of a server hosted in our offices.

2.2.7.2 Evaluation Results

As noted, we have so far received 11 completed evaluations, but expect more to come in over time. We will update this report as that happens.

Table 2 presents the data we collected characterizing our subjects experience in medicine and medical training. In this and all following tables, the bottom two rows show the average and standard deviation of the subject data in the body of the table. Here, the first column is number of years in the medical field, post-bachelors; we had subjects who were relatively freshly out of medical school as well as veterans, with the average approaching at a healthy 14 years of experience. The other five columns requested subject self-assessment experience ratings on a scale of 1 ("not experienced") to 5 ("very experienced") bearing on several aspects of medical practice, education, and training technology.

Essentially all subjects rated themselves highly on experience in emergency medicine. There was more diversity in their experience developing and delivering medical education/training (not always correlating with seniority in the field), and averaging out in the middle of the scale. Interestingly, *use* of computer-based training (CBT) also averaged out in the middle of the scale; it seems that some forms of CBT have become reasonably wide-spread in the medical field. On the other hand, significant experience with *development* of CBT was rare (though half the subjects claimed *some* experience in that area).

Table 3 presents data we collected on the amount of time subjects spent on the evaluation. These values are fairly homogeneous and reflect reasonably well our expectations and advice, with a few notable exceptions (primarily a couple of outliers on the high side for set-up time). The most important information here is that subjects did spend significant time with the simulation, averaging out to just about exactly an hour. We expect that the high numbers for Set-Up (e.g. installation of the software) and Video & Help (e.g. pre-scenario orientation to the software's use) reflect relative inexperience and/or insecurity with computer use (and perhaps difficulty with some of the issues described in the Method section above). In any case, in a real fielded version of METTLE, individual users would not be asked to install their own Servers.

⁶ There was a 10%-15% rate of returned undeliverable solicitation letters due to "bad" addresses.

⁷ This figure includes what appears be a cluster of perhaps half-a-dozen word-of-mouth referrals within one medical school.

Table 2. Subjects' Levels of Experience

Subject	Years in Field	Emerg. Medicine	Develop Training	Deliver Training	Using CBT	Develop CBT
1	24	5	4	4	3	1
2	16	4	2	2	2	1
3	12	5	5	5	3	2
4	30	5	4	4	3	3
5	19	4	1	1	2	1
6		4	4	3	4	2
7	8	5	3	4	2	1
8	6		4	4	4	4
9	1	4	1	1	3	1
10	11	5	3	3	3	3
11		5	2	5	5	1
Mean	14.11	4.60	3.00	3.27	3.09	1.82
Std Dev	9.13	0.52	1.34	1.42	0.94	1.08

Table 3. Time Spent by Each Subject on Evaluation Tasks

Subject	Set-Up	Video & Help	Scenario Run	Write-Up
1	10	15	45	5
2	10	10	45	5
3	10	5	60	10
4	60	30	60	10
5	5	5	50	5
6	15	10	90	10
7		5	60	15
8	10	15	60	5
9	90	5	75	10
10	30	30	60	10
11	25	20	60	10
Mean	26.50	13.64	60.45	8.64
Std Dev	27.59	9.51	12.93	3.23

Table 4 starts to get at the instructional effectiveness of the METTLE approach to medical training. We asked subjects to rate their perceptions of how the approaches illustrated in our prototype were likely to compare on a range of training objectives against more traditional approaches such as attending lecture or reading articles. Again these data are ratings on a 1 to 5 scale, this time ranging from “Much less effective” to “Much more effective” (higher numbers are better). A value of 3 was anchored to “As effective” as these traditional methods of instruction. The four training objectives we surveyed included (a) exposing students to uncommon situations with accompanying knowledge and skills, (b) allowing students to practice applying such knowledge and skills, (c) acquiring proficiency at emergency medical response, and (d) identifying gaps in knowledge and skills.

So far all subjects but one have rated the METTLE simulation as likely to be at least as effective as traditional instructional methods with regard to all four learning objectives. The averages all come out near 4.0 (plus or minus 0.2 or less). *Practicing* use of knowledge and skills and identifying *gaps* come out slightly on the higher side. *Exposing* students to uncommon situations and achieving *proficiency* come out slightly on the lower side. There is some logic to these relative rankings, as a simulation is good for practice, and one side-effect of (critiqued) practice is to recognize where your gaps are. On the

other hand, it may be that readings and lectures are just as good for simply exposing students to odd cases. Finally, we suspect that limitations in the fidelity of the METTLE simulation may lead subjects to hedge about whether it is capable (better?) at helping students achieve proficiency in practice.

Table 4. Subjects' Ratings of Effectiveness of Instructional Method

Subject	Expose to Uncommon Cases	Apply Knowledge & Skills	Gain Proficiency	Identify Gaps
1	3	4	4	4
2				
3	5	5	5	5
4	3	2	2	2
5	4	5	4	5
6	5	4	4	5
7	3	4	3	4
8	4	3	3	3
9	4	4	3	3
10	4	5	5	5
11	4	5	5	5
Mean	3.90	4.10	3.80	4.10
Std Dev	0.74	0.99	1.03	1.10

Table 5 reflects data from a final set of questions intended to dig deeper into the particular aspects of METTLE which might account for different kinds of instructional effectiveness. We asked subjects to rate seven different aspects of the METTLE system, again on a scale of 1 to 5, this time ranging from "Not effective" to "Highly effective" (again, higher numbers are better). The seven system dimensions cover a general assessment of the scenario structure, four specific aspects of the simulation, and two aspects of automated Tutor support. Again ratings cluster around a value of 4.0 (mostly plus or minus 0.2); the system is considered overall effective.

Table 5. Subjects' Ratings of Specific Aspects of Training System

Subject	Scenario Challenges Knowledge & Skill Application	Patient Provides Decision Cues	Chart Organizes Data & Supports Decisions	Other Actors Provide Decision Cues	Simulator Interactions Allow to Demonstrate Competence	Tutor Explores Rationale & Decisions	Tutor Feedback Promotes Learning & IDs Gaps
1	4	3	3	2	2		3
2							
3	5	5	5	5	5	5	5
4	3	3	3	3	4	4	5
5	4	4	4	4	3	5	5
6	5	4	5	5	4	5	4
7	3	2	2	3	3	3	3
8	3	4	4	4	4	4	4
9	4	3	5	3	4	4	4
10	4	5	4	3	4	3	4
11	5	5	5	5	5	5	5
Mean	4.00	3.80	4.00	3.70	3.80	4.22	4.20
Std Dev	0.82	1.03	1.05	1.06	0.92	0.83	0.79

We find it encouraging that the overall scenario structure gets a solid 4.0 rating, and that the two aspects of the Tutor come out on the high side at 4.2. The lowest ratings are associated with areas that

had known limitations. Feedback during our formative evaluation after the second development cycle identified limits on the extent to which a low-fidelity, non-immersive simulation could support cues and actions a doctor might have access to in the real world. In our subsequent design and implementation revisions we were able to flesh out the patient behaviors fairly well within those limitations (though clearly we could still do more, even in a web-delivery context, e.g. in terms of adding sounds and more useful visuals to the patient examination). However, we recognized that we were not able to push as far on refining the universe of student actions beyond the diagnosis phase. Likewise, we believe we could push further on the extent to which interactions with other actors are fleshed out (currently our lowest score at 3.7).

Figure 12 summarizes all of the METTLE ratings data presented above as a series of ratings value bar charts to emphasize the distribution of subject responses. The top row of four graphs shows the *relative METTLE effectiveness* ratings. The bottom two rows (comprising seven graphs) show the *specific METTLE feature effectiveness* ratings. The clustering of most values around a rating of 4 is visually apparent. The skewing of the Tutor ratings (the last two graphs in the bottom row) to the high side is also suggested.

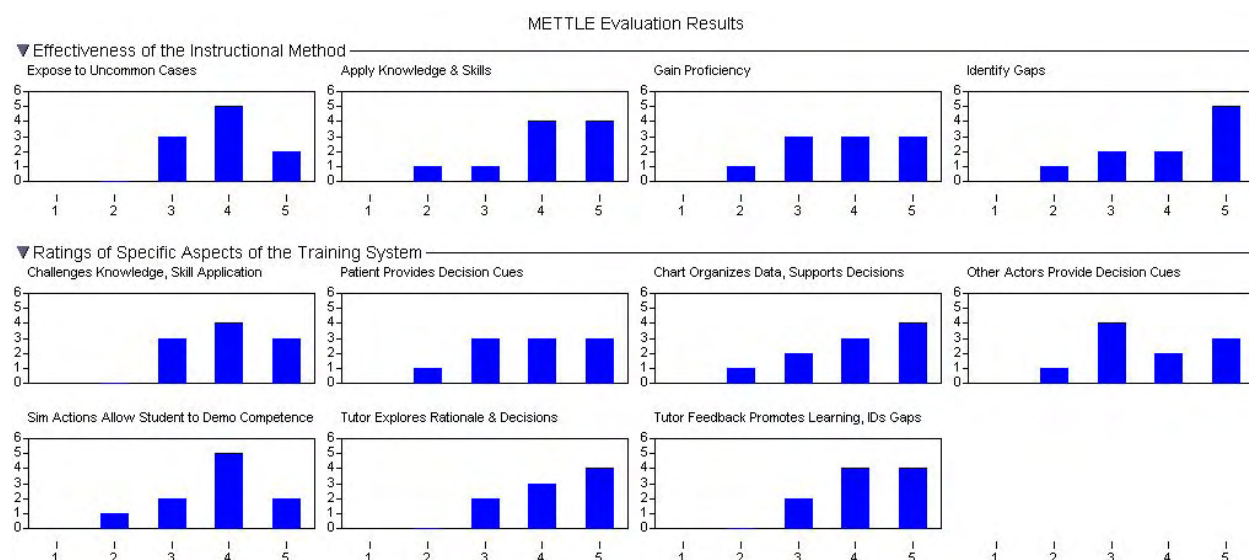


Figure 12. METTLE Evaluation Ratings Distributions.

2.2.8 Cost/Benefit Analysis

In this section we review and synthesize what we have been able to determine about the possible costs and benefits of a deployed METTLE system. The major costs include (1) development, (2) delivery, (3) support, and (4) student time. The major benefits include (1) improved student knowledge, skills, and task performance, (2) avoided costs. We start with costs, which are the best quantified.

- Scenario Development Costs:** In Section 2.2.6, as we described our proposed authoring methodology, we also tracked associated costs on per-scenario basis. Our estimates came out to between \$30,000 and \$45,000 per hour of scenario playing time. While this is high in absolute terms for training development, it is useful to consider the investment scaled against the size of the potential student population and the value of their time. The U.S. has on the order of 32,000 emergency care physicians, and approximately seven times as many physicians in primary care. Thus there are thus at least 250,000 physicians in our medical system who could benefit from continuing education of the sort METTLE can offer. Focusing slightly differently, there were on the order of 15,000 students in the graduating medical school class of 2007. There is certainly room for METTLE scenarios to server a thousand (or many thousands), of students *per year*,

which makes the amortized development-cost-per-play of a scenario that maintains its utility for several years small to negligible compared to the costs of the trainees' time. We will use \$10/hour of training as our estimate (1000 students/year for a 3-5 year scenario lifetime).

- **Scenario Delivery Costs:** Looking at delivery costs, we are in the web realm of vanishing marginal costs for electronic media distribution. However, it is still worth estimating what those costs might be. Consider that a one hour scenario playing time might only involve a couple of hundred web pages requests—we would be on safe ground to estimate one get or post operation every 10 seconds. Ignoring network latency, all METTLE requests to the server complete in well under a second. Supporting an average of 10 simultaneous users on a normal desktop machine should not be any trick at all, even with relatively spiky load distribution. To stay conservative, we will consider only a ten-hour day, meaning at the low end a cheap machine should easily support delivery of 100 training hours/day. At this level \$1000/year for hardware is a good estimate. \$1000/year for a suitable network connection is also plausible, as METTLE pages tend to run around 25K-Bytes, plus (non-cached) media ranging say between 50K-Bytes to a maximum of about 200K-Bytes for a long sound clip; given that many round-trips to not involve any new media, an estimate of 100K-Bytes per page is probably generous, which at our nominal user load would lead to a need for an average 100K-Bytes/sec of bandwidth. In the end, the support costs for the machine and network connection would likely dominate delivery costs, and would depend on whether there was a support organization in place for other purposes. To stay with round numbers, we estimate a total of \$10,000/year in delivery costs to serve 100 hours of training/day and 300 working days/year.⁸ At these (very small and inefficient) scales, training delivery costs come to \$0.33/hour of training.
- **User Support Costs:** We believe user support costs should be comparable to delivery costs. In its current prototype release, the only major technical difficulties associated with getting METTLE running on a browser client are (a) making sure users are running a recent vintage browser, and (b) making sure they have pop-up blocking turned off. Having identified these issues, it should not be difficult to build in tests for problems, and advice for how to fix them. Conceptual difficulties (e.g. helping students understand what the system is supposed to do and how it is supposed to be used) seem to have been reasonably well addressed by our existing introductory video. Producing a more polished video, and providing the additional network bandwidth required for all prospective users to download it are small additional costs. If our 30,000 scenario plays/year represent as many as 10,000 distinct users, each downloading a 50M-Byte video once, we may double our server and network loads. In addition, we can provide an email address where users can address queries and problem reports. In all, we estimate an addition \$10,000/year for these support costs, or another \$0.33/hour of training.
- **User Time:** At the scales we have been discussing, we project training costs of around \$11/hour. Certainly for practicing physicians, the costs of their time dominate the costs of the training itself. While that is nice, in terms of making the technology costs seem reasonable, it only makes things worse if it turns out that the technology is not making good use of students' time. If our arguments so far are correct, the dominant question becomes a comparative one of how much is learned in an hour of METTLE scenario play, as compared to an hour spent doing something else like reading or attending lectures. Hard data on such issues is always difficult to come by, especially when (1) there may be disagreement about exactly what people are supposed to be learning, (2) when different training regimens naturally emphasize (and value) different kinds of training outcomes, or (3) when what is easy to measure as an outcome does not align well with what a technique claims to teach.

⁸ Note that our Delivery Cost estimates end up assuming 30,000 hours of delivered instruction/year. If we really only have 1000 students/year for each 1-hour scenario, then we would need 30 scenarios to drive that much use.

Assessing the costs of user time in a comparative sense has already raised the question of METTLE's distinctive training benefits. We briefly mention two:

- **Improved student knowledge, skills, and task performance:** Since the bottom line in medicine is often life and death—and in the case of CBR contingencies, life and death on a massive scale—it is not hard to claim potentially huge benefits from any improvement. We cannot really do expected-value calculations because we do not know the probability of the events for which we seek to prepare physicians—we know it is low for any given doctor, but we also have good reason to believe it is certainly non-zero for *some* doctors. So rather than try to argue for any quantified expected value of METTLE training, it seems best to fall back on the same kinds of assessments already suggested by discussion of the (relative) costs of user time. The expected benefits of METTLE-style training should lie in (1) increased motivation and attention leading to increased retention, (2) learning through use leading to increased ability to use what is learned (avoidance of *inert knowledge*), (3) embedded adaptive guidance and critiques leading to more appropriate and accurate learning. The ultimate METTLE training bake-off would depend on development of measures of appropriate, flexible, efficient knowledge application in coping with CBR contingencies.
- **Avoided Costs:** We can be on slightly more solid ground when we turn to the question of costs that METTLE may help students to avoid. The most obvious such costs, often cited in support of distance learning, are (1) the costs associated with travel to an educational site, and (2) the inefficiencies of being constrained to do the training at times that may not be convenient. These arguments do not apply when the alternate form of training is to read books or papers. Further, these costs may be minimal for students who are actually students—regularly attending an educational institution, with no higher priority than to go to regularly scheduled educational events. However, the costs quickly mount for working professionals when the alternative is attendance at a seminar or meeting.⁹ It doesn't take many minutes of commute-time or down-time around a scheduled meeting to consume \$11 of professional time.

In advance of agreed-upon training efficacy metrics used as the basis for large-scale comparative training method evaluation, we have the subjective judgments of a growing sample of emergency physicians who believe METTLE provides overall effective training that they further generally believe to be more effective than reading or lectures.

Economically speaking, there is a case to be made for METTLE. If a low-cost scenario distributor could capture a user population of 1,000-10,000 users/year, each playing between 30 and 3 1-hour scenarios, and willing to value the training at \$20/hour there is the beginnings of a business. A much smaller launch of 1000 users/year each playing 3 1-hour scenarios willing to pay \$30/hour could perhaps work as well. An alternate business model could be built on bulk sales of semi-custom scenarios for large medical organizations (e.g. commercial HMOs and hospital chains, TRICARE, DMRTI, or USUHS). Selling to organizations with hundreds or thousands of physicians could lower sales costs or possibly raise per-user value since scenarios could be tailored to the specific needs and circumstances of each medical organization.

⁹ Though even there, subsidized travel to desirable locations may sometimes cause this argument to cut the other way. Some professionals enjoy the travel.

3 Key Research Accomplishments

Our major research accomplishments include:

- We have developed techniques for scripting *non-player characters* in a medical training scenario including patients, supporting medical staff, and an automated Tutor.
- We have implemented two different technologies for supporting *conversational agents* appropriate to the two significantly different contexts of *student-initiative dialog* and *agent-initiative dialog*.
- We have refined our end-user runtime system enabling it to support a range of interaction styles in a pure web delivery format: in addition to language-based interactions (including recorded speech output) we support *interactive images*, a multi-page *tabular and graphical patient chart*, *order forms*, and *hierarchically organized choices* in extended dialogs.
- We have developed inventories of reusable patient behavior cues, including approximately 300 diagnostic interview questions as well forms for ordering a useful range of tests and treatments.
- We have developed conventions and examples of using the agent-initiative dialog infrastructure to construct *Socratic dialogs* that explore student's diagnostic rationale.
- We have developed tools to ease the authoring of simulated agent behaviors by enabling the use of COTS spreadsheet applications to easily specify most of the required behaviors.
- We have developed other web-based tools to support basic authoring of course level components such as curricula and high-level scenario structures.
- We have developed a complete example training scenario for ED physicians enabling them to learn diagnostic, treatment, and response principles for recognizing and dealing with anthrax.
- We have carried out an evaluation study of our final product which provides suggestive evidence that the system and scenario are in fact technologically solid, educationally appropriate, and pedagogically effective.
- Finally, though outside the scope of this project's funding, we have already begun to demonstrate that the infrastructure built for METTLE can be applied to other kinds of professional level decision making and team collaboration training applications of interest to the U.S. Army and wider government circles.

4 Reportable Outcomes

The major reportable outcome of this project is the METTLE software and this project Final Report. A book chapter describing METTLE is also being prepared for publication.

5 Conclusions

Our work on the METTLE project ranged widely covering (1) training needs and domain analysis, (2) simulation and tutoring technology (including packaging for reuse and packaging for web delivery), (3) domain, scenario, and instructional content representation, (4) a multi-layered approach to tools for speeding content authoring, and finally (5) evaluation and analysis of the results.

With regard to training needs and domain analysis, we identified an important community that was being underserved with respect to CBR training: emergency department, and eventually primary care physicians. We identified a set of CBR-related issues on which this community needs training, and for which METTLE's discussion-centric decision-making training is appropriate. Especially when it comes to more clandestine attacks with delayed consequences, the diagnosis, treatment, and response decision-making of these front-line physicians is likely to determine how well damage can be contained. We developed an inventory of CBR agents of concern, appropriate to framing diagnostic differentials and defining a space of potentially useful scenarios. We also developed preliminary taxonomies of other key domain areas such as relevant tests, findings, and treatments. Finally, we identified an appropriate range of interview questions useful across a wide range of diagnostic encounters. All of this was required to shape the rest of the project work.

In the area of simulation and tutoring technology, the focus that emerged in our training goals helped us identify a specific range of required simulator support, and let us to focus on straightforward discrete event simulation, dominated by agent behavior and language-based interaction simulation. We designed a unified framework for scripting agent behaviors, and built within that framework two different dialog support mechanisms appropriate to student-initiative and agent-initiative dialogs, each of which have their place in a free-play instructional system (most frequently student-initiative for free-play, agent-initiative for instruction). Within this same framework, we developed techniques for integrating Tutor behaviors with the underlying world simulation. Further, we explored two major approaches to unifying the simulation machinery with a student User Interface (UI): a configuration-based approach emphasizing reuse of components and easier assembly of alternate UIs, and a web-based approach emphasizing delivery of a richly interactive user interface through a standard web browser. We are strongly encouraged by the results to date in the areas of scenario-scripted language interaction and web-based delivery of interactive simulations, and expect to pursue elaborations of both of these in future projects.

To make the simulation machinery run, we defined supporting data representations for curriculum, domain concepts, scenarios, agent behaviors (including several classes of Tutor behaviors), extended dialogs, and interaction mechanisms (e.g. live images and forms). Especially as we moved to web-based delivery, we emphasized use of conventional media formats to the greatest extent possible, and adopted the convention that all system data should have a human-readable textual form enabling direct inspection and editing. Using these representation formats we built encodings for the various conceptual spaces identified through domain analysis: CBR agents and other conventional medical conditions, diagnostic tests, possible findings, and relevant treatments. We built a basic inventory of behaviors, including the range of diagnostic interview questions. We built a range of supporting interactive order forms. All of this means we are prepared to develop additional relevant training scenarios and can do so more efficiently by leveraging pre-existing content.

Our decisions about data representation, our work with web-based UIs, and our experience building and using earlier generations of authoring tools all fed into decisions about authoring techniques. First we do not believe there is any one-size-fits-all approach to authoring. Tools appropriate to one class of user may be annoying and frustrating (or incomprehensible) to another class of user. Further, putting together a complete scenario requires a range of skills, most likely embodied in a range of individuals. When it comes to developing custom authoring tools costs—both obvious and subtle—have to be recognized. There are the obvious development costs of building such tools. But there are also less obvious costs of delay and frustration as early versions of the tools do not quite measure up (e.g. arrive late, lack features,

or turn out to have missed key requirements), as later releases continue to have issues (e.g. lack of stability, scalability, or robustness relative to mainstream commercial applications), and as the custom authoring suite struggles to keep up with evolution in the underlying models of what is being authored. These factors are especially noticeable when it comes to authoring of high volumes of content (e.g. hundreds of behavior script lines).

Such considerations motivated our experiments with exploiting COTS tools for authoring. Our experience suggests that for any given form of data, it is effective to (a) define a simple inspectable textual format, (b) build code for loading and validating data in that format into the runtime system, (c) then define translators allowing transformation of data from alternate (COTS tool) formats into the underlying text format for import and validation, (d) if necessary prepare templates, style sheets, macros or other customizations of the COTS tools, (e) set up the environment so there is a smooth cycle of COTS editing, translation, import, and validation making it easy catch errors introduced because of this looser form of authoring support, and finally (f) in the rare case where performance becomes an issue add a compiler to pre-process the textual form in ‘a’ for more efficient loading into the runtime system. It is far easier to build robust and efficient text-to-text translators and validators than to build UIs that compete with the hundreds to thousands of person-years invested in major COTS applications, not to mention the millions of hours of cumulative user experience and testing. As with dialog scripting and web interfaces, this is a direction we expect to pursue further in the context of future projects.

Finally, our evaluation of the final Phase II METTLE prototype has been very encouraging. A wide range of emergency physicians (in age, gender, location, experience), subject to all the time pressures of such professional life, and offered very little support, were able to install and run the system, and for the most part enjoy the experience and judge it to have substantial educational potential. As is common in ITS projects, a lingering question is the extent to which the potential benefits balance with the expected costs. We believe we have quantified the costs of developing, delivering, and maintaining METTLE scenarios. We have estimates of the potential user population that could benefit from METTLE training. And we have identified some of the common alternate training mechanisms with which METTLE would compete. Taking further steps in refining this picture will fall into area of Phase III development.

The ultimate payoff of this line of work should be to provide medical professionals, including those in the DoD, with training that goes several steps beyond the facts and procedures that they can read about in books and papers. Medical competence is not just about what you know; it is about what you can do with what you know. Simulation-based training can provide a compelling, motivating, and memorable context not only for presentation of medical knowledge, but also for practice of medical skills. But simulation-based training is most effective when a coach is on hand to shape the experience and help draw out the lessons. METTLE offers added value by making such an automated coach available to all students—anywhere, any time. The first contribution of this work, then, is to help enrich medical training, moving it from traditional media, to simulation-based training, to simulation-based ITS.

METTLE, however, introduces tutoring capabilities beyond those that are becoming somewhat common across a range of simulation-based ITSs. Medical decision-making skills are not simply a matter of rote application of practiced behavior. The essence of professional skill is judgment based on experience. It is no accident that Socratic instruction is so often associated with professional level training, as in law, business, and medical schools. METTLE delivers novel Socratic tutoring technology so that the embedded coach can interactively explore the rationale underlying students’ decisions—can help them learn to reason better about the application of their medical knowledge and skills.

Finally, medical practice is not only about what the individual care provider can do, but about what they can do when interacting with others. METTLE offers technology to simulate interaction with patients and with an extended medical team. Just as medicine cannot be practiced in a world without other people, so too some of the lessons METTLE teaches could not be delivered in a simulated world that did not include a cast of such characters.

6 References

- ACEP (2001). *Developing Objective, Content, and Competencies for the Training of Emergency Medical Technicians, Emergency Physicians, and Emergency Nurses to Care for Casualties Resulting From Nuclear, Biological, or Chemical (NBC) Incidents: Final Report*. American College of Emergency Physicians—NBC Task Force. Available on-line at www.facs.org/trauma/nbcreportacep.pdf
- Anderson, J.R., Boyle, C., and Reiser, B. (1985). Intelligent Tutoring Systems, *Science*, **228**:456-462.
- Bickley, L.S. (2003). *Bates' Pocket Guide to Physical Examination and History Taking*. Lippincott Williams & Wilkin.
- DMRTI (2003). Chemical, Biological, Radiological, Nuclear, and (High Yield) Explosives (CBRNE) Training – Standards of Proficiency and Metrics.
- Deerswester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., & Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, **41**, 39-407.
- Domeshek, E, Holman, E. & Ross, K. (2002). Automated Socratic Tutors for High Level Command Skills. Proceedings of the 2002 Interservice/Industry Training Simulation and Education Conference.
- FM 8-284 (2000). *Treatment of Biological Agent Warfare Casualties*. Headquarters, Departments of The Army, the Navy, and the Air Force, and Commandant, Marine Corps. Available on line at <http://www.globalsecurity.org/wmd/library/policy/army/fm/8-284/fm8-284.pdf>
- Graesser, A.C., Person, N.K, Harter, D., and the Tutoring Research Group (2001). Teaching Tactics and Dialog in Autotutor. *International Journal of Artificial Intelligence in Education*.
- Jernigan, J.A., Stephens, D.S., Ashford, D.A., et al. (2001). Bioterrorism-Related Inhalational Anthrax: The First 10 Cases Reported in the United States. *Emerging Infectious Diseases*, **7**(6). On line at <http://www.cdc.gov/ncidod/EID/vol7no6/jernigan.htm>
- LeBlond, R., DeGowin, R., & Brown, D. (2004). *DeGowin's Diagnostic Examination*. McGraw-Hill.
- MMOAFRRI (2003). *Medical Management of Radiological Casualties Handbook (2nd Ed)*. Military Medical Operations Armed Forces Radiobiology Research Institute, Bethesda, MD. On line at <http://stinet.dtic.mil/cgi-bin/GetTRDoc?AD=ADA415842&Location=U2&doc=GetTRDoc.pdf>
- Munro, A. & Pizzini, Q.A. (1995), RIDES Reference Manual, Los Angeles: Behavioral Technology Laboratories, University of Southern California.
- Ong, J. & Ramachandran, S.(2002). Intelligent Tutoring Systems: The What and the How. *Learning Circuits*, February, 2000. Available at <http://www.learningcircuits.org/feb2000/ong.html>
- OSG (1997). *Medical Aspects of Chemical and Biological Warfare*. Borden Institute, Textbooks of Military Medicine. Available on line at http://www.bordeninstitute.army.mil/published_volumes/chemBio/chembio.html
- Porter, M.F. (1980). An algorithm for suffix stripping, *Program*, **14**(3) pp 130–137.
- Recker, M., Ram, A., Shikano, T., Li, G., and Stasko, J. (1995). Cognitive Media Types for Multimedia Information Access. *Journal of Educational Multimedia and Hypermedia*, **4**(3), 183-210.
- Seiple, C. (1997). Consequence Management: Domestic Response to Weapons of Mass Destruction. *PARAMETERS, US Army War College Quarterly*, Autumn.
- Smith, R. (2003). The Application of Existing Simulation Systems to Emerging Homeland Security Training Needs. In Proceedings of *Simulation Interoperability Workshop—Europe 2003*.

- Temte, J.L., and Zinkel, A.R. (2004). The Primary Care Differential Diagnosis of Inhalational Anthrax. *Ann Fam Med.* 2(5): 438–444. On line at <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=1466714>
- USAMRICD (2000). *Medical Management of Chemical Casualties Handbook (3rd Ed)*. U.S. Army Medical Research Institute of Chemical Defense (USAMRICD), Aberdeen Proving Ground, MD
- USAMRIID (2005). *Medical Management of Biological Casualties Handbook (6th Ed)*. U.S. Army Medical Research Institute of Chemical Defense (USAMRICD), Ft. Detrick, Frederick, MD. On line at <http://www.usamriid.army.mil/education/bluebookpdf/USAMRIID%20BlueBook%206th%20Edition%20-%20Sep%202006.pdf>

7 Appendix A: Project Activities

This appendix largely summarizes and organizes material that has already appeared in interim project reports. All activities are included here for completeness. The Phase II work plan called out twelve tasks. Table 6 lists those tasks. We use it as an organizer for reviewing our project activities. The tasks break down into three groups: (1-4) up-front tasks that we addressed primarily during the first year of the project, (5-9) development tasks that we carried out in three major waves, and (10-12) evaluation, analysis, and reporting tasks culminating in this report.

Table 6. Proposed Task List with Capsule Descriptions

	Task	Description
1	Find End-Users	Seek participating end-user organizations
2	Requirements	Refine system requirements
3	Scenario Definition	Define appropriate training scenarios
4	Design Refinement	Refine overall system design
5	Simulation Integration	Acquire and/or build simulation engines as needed Develop integrated simulation environment
6	Runtime Development	Develop scenario in which students play scenarios Emphasis on supporting language-based interaction
7	Instructional Model	Define formats and generate knowledge structures Develop common framework and processing/control
8	Sim/GUI Configuration	Generate/package reusable components in framework Adapt system for easy (web-based) deployability
9	Authoring by Example	Develop authoring approaches to speed development Develop authoring tools to make development easier
10	Evaluation	Gather feedback on successive system releases Conduct Final Evaluation on deliverable system version
11	Costs/Benefits	Identify costs associated with system development/use Identify likely benefits associated with system use
12	Documentation	Quarterly, Year-1, and Final Reports Presentations at reviews and meetings Published articles

1. Seek Participating End-User Organizations. Seeking a better understanding, and soliciting input from representatives of possible METTLE user communities was an active task from Phase I through the first year of Phase II. At the Phase II kickoff meeting we discussed the issue of whether to continue focusing on hospital settings or to shift towards field (first-responder) settings. The 91W School at Ft. Sam Houston, TX was identified as a likely teaming prospect, and TATRC agreed to help establish initial contact. We carried out some background research into the first-responders' context for CBRNE events. We studied materials on CBRNE response directed towards first-responders. We reviewed material on the civilian incident command system, including its ties to the military. Unfortunately, efforts to contact the 91W School did not mature during the first quarter of the project, so as discussed at the kickoff, we moved forward with initial Phase II development work based on our Phase I hospital emergency department setting.

Through the good offices of TATRC, Dr. Domeshek was able to attend the U.S. Army 91W Trainers Conference in San Antonio, TX from 10-12 May 2005. The purpose of this trip was to learn more about the training needs of Army medics and to explore possible fits between the METTLE project and the 91W community. In addition, the trip afforded an opportunity to see some of the cutting edge forms of simulation that the Army and others are developing for emergency medical personnel. Overall the trip

was interesting and informative, but did not suggest any obvious avenues for interaction. The primary concern at the conference was trauma medicine; CBR or WMD incidents were never mentioned. Conversations with attendees attempted to explore the relevance of CBR training for medics and produced limited, but potentially useful results:

- 91W training does not seem to include any special training on CBR issues.
- The military has special schools/courses for CBR medical response (e.g. offered by the U.S. Army Medical Research Institute of Chemical Defense, or the U.S. Army Medical Research Institute of Infectious Diseases).
- Civilian EMT training does not yet include formal standardized CBR or WMD material because the National Registry of EMTs last revised their standards pre-9/11. The NREMT were scheduled to start revising their EMT-B standards in October 2005, and will probably finish in 2008. After that, it is likely that WMD training will become a more serious and standardized requirement for EMTs (which will probably extend to 91Ws).
- There are several companies that offer distance learning of various kinds for EMTs. These companies could act as publishers (distributors, marketers) for modules produced using a system such as METTLE. This will become a more likely channel after the NREMT starts to require training on these topics.

As a result we continued work on developing the METTLE system with its initial hospital-based anthrax scenario, and began to seek out a wider circle of domain experts in the local area, largely in order to prepare for the initial formative evaluation, but also as a way to better understand the professional communities we hope to serve. As described under Task 11 below, the formative evaluation did end up producing some useful insights into possible user communities and commercialization pathways.

2. Refine System Requirements. We believe we identified a solid set of requirements during our Phase I work. Activity on this task during Phase II was primarily aimed at incorporating any constraints identified as we learned more about potential user communities, and remaining aware of trends or changes that may affect our work (e.g. new applicable or competing technologies; coming changes in the context or needs of user communities).

Early on, we explored options to address the identified user desire to have speech-based input and output, experimenting with approaches to such technology that would also preserve our ability to run in a text-based mode. We ultimately concluded that COTS speech understanding systems offered the best performance and interoperated well with leading web browsers which became our target client delivery platform. Generic (e.g. non-medical) versions of these products suffice for the majority of our speech input needs, and are licensed at quite reasonable rates on a per-desktop basis. Since an affordable desktop technology that requires per-speaker training proved to be the leading and adequate solution, we decided to leave it to individual users to choose, install, and use speech input technology or not at their option. With regard to speech output, we decided to take advantage of web browsers' ability to play standard audio formats, and serve pre-recorded speech files along with simulation web pages.

While attending the MMVR conference (26-29 January 2005), we made special note of those projects being reported that focused on CBRNE response. There are a number of CBRNE-related projects under way that focus on first-responders; however, essentially none seem to take emergency CBRNE hospital care as their focus. We took this as a promising indicator that there was a niche we could fill by continuing to focus on the hospital-based setting. On the other hand, there was at least one project that looks at hospital emergency teamwork (an important goal for METTLE), though focusing on distributed training using avatars of live trainees, rather than light-weight simulations of other team members.

On 21 March 2005, Ray Perez of ONR visited Stottler Henke's Boston-area office to discuss the METTLE project and its possible relationship to a web-based information management project at Aptima, Inc. Dr. Domeshek spent about two hours discussing the U.S. military's re-configuration of its medical

services as expeditionary forces, and the information dissemination and training issues that follow from this trend.

During the end of September and the beginning of October 2006 we conducted our own internal review of the results from our first round of Phase II METTLE development. Project team members and other Stottler Henke employees were shown various pieces of the system and provided feedback, which was later organized to supplement prior requirements and designs.

3. Define Appropriate Training Scenarios. Our target for the Phase II project was two major training scenarios. In the end, we found there was adequate challenge in a single scenario, and our focus on simulating interaction towards diagnosis and treatment of a patient with anthrax served as an effective way to drive much of our project work. The anthrax scenario presented a range of challenges that led us to our ultimate overall system architecture (described in Section 2.2.2), shaped our final runtime interface (Section 2.2.3), and drove our approach to authoring (Section 2.2.5), while motivating and helping to constrain our explorations of key aspects of knowledge and action in the bio-medical domain.

During the project's first year, we brought on an anthropologist, Dr. Kate Gilbert, as a consultant to assist us in the medical domain research required for system development. She worked primarily on the patient-focused phase of the anthrax scenario, with attention to establishing the range of syndromes that might plausibly appear in a student's diagnostic differential, the signs, symptoms, and tests that ought to be pursued and how they could be interpreted, and the possible treatments that would be suggested at different stages in the diagnosis. In addition to discussions with our consulting SME Dr. Workman, she reviewed all available on-line literature about inhalation anthrax cases, and about how doctors recognize anthrax (as opposed to other diseases). She then worked on fitting such data on anthrax into the larger picture of standard medical diagnostic practices, based in large part on a detailed analysis of some standard reference works on diagnostic examinations (LeBlond, Degowin, & Brown, 2004; Bickley, 2003). In addition, she carried out some analysis of the medical management issues that follow upon the diagnosis of a first case of anthrax, and we explored means by which a student in the role of an emergency room physician could be reasonably be introduced to those issues, and how they could play out in an interactive scenario (e.g. using simulated characters to carry some of the tutorial burden in a Socratic format).

Her work contributed to our early explorations of possible scope and form of background medical domain knowledge that would be of use within the system. This in turn motivated explorations of resources available as part of the NIMH Unified Medical Language System (UMLS), including a high-level ontology whose adaptation for use in METTLE we explored. Ultimately we determined that the lexical and conceptual resources in the UMLS were not appropriate to METTLE's needs. They were overwhelmingly large (covering far more than was relevant to our purposes), and not that usefully organized (in particular, with only weak connections between lexical elements and conceptual taxonomies). Having established the need to generate our own more controlled conceptual model, as we fleshed out more of the scenario content (keeping pace with evolving system capabilities, see below), we built concept spaces derived from Dr. Gilbert's research. We enumerated medical conditions, tests, treatments, patient management options, and so on, developing organizational taxonomies and display forms reflecting that organization.

Again, as system capability matured, we also developed media to support the scenario, including patient examination images, recorded patient utterances, and most importantly, patient test results. Finally we developed the actual scenario scripts through several iterations (matching the software design and development cycles), experimenting with different approaches to behavior representation and authoring. Out of this work evolved conventions for structuring scenarios, agent behavior scripts, and Socratic dialogs to explore student's diagnostic rationale.

4. Refine Overall System Design. There was substantial design activity throughout the first year of the project, which tapered off somewhat in later years. The outcome of that design work is described thoroughly in the Results section below. Designs ranged from the general systems framework to specific

modules within that framework. Some of the major elements conceptualized and designed included: (a) the “METTLE Machine” an abstraction layer to unify interpretation of scenario scripts (later renamed the Enact layer 8.1), (b) the “Dialog Package” (later renamed the Discuss layer 8.2), (c) a general instruction management package (the GRAIN layer 8.3), (d) a unifying Interpreter layer (8.4), (e) various components of the user interface (the Runtime 2.2.3), (f) and various authoring tools including web-based tools (10.2) and COTS application based tools (10.3).

5. Simulation Integration. To the extent that this task was intended to address the integration of externally developed simulations, integration turned out to be less of an issue in this project than we had originally supposed. There are two main reasons for this: (1) the highest priority simulation issues have all proven either easily or necessarily addressable within the context of in-house code (thus substantially easing potential integration problems), and (2) the most interesting candidates for external simulator components have proven prohibitive in terms of licensing or technical requirements, at least for now.

To unpack the first observation, consider that the most critical simulation problem faced in METTLE has to do with enabling non-player characters to converse appropriately—that is, in a way both true to their context and useful for training purposes. There is no readily available COTS technology to meet that need, and it is a major focus of our design and development efforts to build adequate tools to that end. Second, consider that the next most basic simulation need is a framework to manage the passage of time and tracking of events (histories and related state changes). This need could be easily addressed e.g. by combining in-house discrete event simulators, with the METTLE-specific scenario script interpreters.

For our application, the areas where external simulators seemed most likely to prove valuable were (a) simulation of individual patient physiological states, disease processes, and test results, and (b) simulation of epidemiological and sociological population effects (large-scale disease spread and management effects, population panic, movement, and infrastructure effects). Driven by TATRC’s guidance to focus on *medical* issues over incident management issues and logistics, and by the need to focus our work to make headway, we have dismissed concerns with simulators of type ‘b’ for now. Our quest for simulators of type ‘a’ did not prove fruitful. Based on feedback from talking to company representatives at the 91W Training Conference trade show floor, companies like Laerdal which may have such disease process models in-house do not seem anxious to license them to small training providers, and the technology may come with odd constraints, such as operating only in the Macintosh environment. Discussion with SimQuest suggested that a patient model of the kind we were talking about would need to be developed from scratch and would be a sizable expense, just to cover one disease, such as anthrax.

As a result, we moved forward with an approach that did not require a robust generative model of diseases processes. We were comfortable, based on the existing example of paper-based case studies and medical school seminar discussions, that high fidelity patient state tracking was not necessary to achieve our training objectives. METTLE’s training scenarios were always intended to be scripted in such a way as to constrain the envelope of possibilities to those that were deemed pedagogically useful. It was never the intent that a student should be able to re-run a scenario a large number of times in order to establish the “best” set of responses (as determined by outcomes in a fully validated physiological simulator). Rather the point was to expose the student to issues, make them seek and apply knowledge in context, interact with teammates to gather information, explore options, and allocate tasks, and promote reflection on appropriate and inappropriate decisions and actions. We believe this can all be accomplished while limiting the range and fidelity of individual patient physiology simulation to essentially what can be described in an augmented case record.

Our approach to developing an integrated simulation evolved over the course of the project’s three planned development cycles. Our first cycle simply extended the Phase I code base as we experimented with system features, look and feel, scenario structures, and early authoring ideas. The second cycle was shaped by exploitation of our in-house SimVentive simulation framework which unifies sub-system and component interactions through a shared event system; this cycle also saw introduction of the METTLE

machine concept. The third cycle was shaped by promotion of the METTLE machine (now the Enact layer) and its underlying Interpreter layer as the unifying mechanism for orchestrating the pieces of the overall simulation.

6. Runtime Development. Work on this task tracked the progression just described for simulation development and integration: three cycles of experimental development moving from legacy Phase I code, to the SimVentive rapid prototyping environment, to the final web-server based configuration with the layered architecture described in Section 2.2.2. Each cycle built on the one before by carrying over successful system architecture, module functionality, scenario structure, and data format designs while making changes driven by evaluations of prior cycles, requirements priority shifts, and the demands of expanding functionality.

Major visible changes along the way included (a) demotion of time overt tracking and management, and elimination of spatial mapping with its location and explicit tool metaphors, (b) promotion of the tutor to co-equal prominence in the display layout, substituting for the patient chart which was demoted to pop-up window status, (c) splitting of patient examination interactions out from the base patient interactor and chart, and introduction of a set of other special-purpose form-based interactors, e.g. for ordering medical tests and treatments, (d) introduction of mechanisms for requesting Tutor hinting/explanatory comments, and (e) introduction of a Reference page for looking of useful information during the scenario.

7. Instructional Model Generalization. After ensuring some mechanism for simulating relevant aspects of a world (in this case emphasizing language based interaction with simulated actors, including patients), and packaging those simulators for effective use by a student, the move from *game* to *training system* requires introducing appropriate adaptive instructional interventions. Work on the tutoring aspects of METTLE really started with the introduction of the METTLE Machine (an extensible framework for multi-agent simulation including basic language-based interaction) and the Dialog Package (a set of tools specifically designed to support more complex and extended agent-initiative dialogs) during the second development cycle. The METTLE Machine's rationalization of simulated agent behaviors made it easier to track meaningful student actions and provided a common locus for attaching tutoring knowledge. Likewise the construction of an agent-initiative dialog system leveraging ideas from earlier projects aimed at Socratic tutoring provided infrastructure for extended tutoring interactions appropriate to training decision-making through rationale exploration.

This work continued during the third development cycle, as several distinct modes of tutor behavior were elaborated and supporting annotation types introduced on character behaviors. Proactive prompting, reactive feedback, and hinting/explanatory comments were differentiated, and mechanisms introduced to encode and control them, including condition-testing, time-delays, and priority ranking. Likewise work on Socratic dialog infrastructure and conventions of use was a major part of the third development cycle. During this time, the Discuss layer (8.2) was extended to exploit the Concept layer (8.5) both for potential user response generation and for condition testing.

8. Simulation/User-Interface Configuration Generalization. Work in the general area of User Interface (UI) infrastructure as it relates to simulation and training scenario development went through two major phases during this project. During the second development cycle we focused on our originally stated goal of emphasizing increased configurability to speed authoring. However, during the third cycle, largely in response to experience and feedback suggesting light-weight web-based delivery of METTLE training would be a valuable feature in keeping with the project's stated deployability goal, we shifted focus to emphasize achieving our target runtime interaction using techniques appropriate to unaugmented mainstream web browsers. This shift was in keeping with project goals, as *deployability* was a key theme in the original solicitation, and the *configurability* task was explicitly called out in the proposal as having relatively lower priority than other goals.

During the second cycle, then, activity related to this task focused on use and extension of our in-house SimVentive simulation framework. SimVentive, developed by Stottler Henke for the U.S. Air Force to support simulation gaming, was designed with an emphasis on extensibility and configurability. In addition, it integrated Stottler Henke's GRIST representation framework, which METTLE was already

intending to use for capturing domain knowledge. During the course of integrating other second-cycle developments with SimVentive (e.g. the METTLE Machine and Dialog Package) and using SimVentive to construct the refined Runtime UI, the METTLE project made several enhancements to SimVentive (reported in Section 2.2.1.1) as well as (on the authoring side) related contributions to the configurability of GRIST's editing environment (reported in Section 2.2.1.2).

During the third cycle, activity related to this task focused on translating METTLE's UI to the web environment, adopting the constraint that we use only standard techniques and data formats that would not require special browser plug-ins: HTML, CSS, JavaScript, images, and sound files. The goals were to minimize requirements for specially configured or high-powered client machines, to avoid version incompatibilities, start-up, or performance delays associated with mobile code downloads, and to promote central administration of a METTLE deployments, easing updates to code and scenario libraries. The result was a reconceptualized METTLE system architecture that layered the major simulation and tutoring processing elements designed in earlier cycles over a web servlet architecture (see Section 2.2.2).

9. Scenario Authoring by Example. While cost management through enhanced authorability was a major theme of the project, as with *configurability* above, the specific idea of *authoring by example* was explicitly called out in the proposal as having relatively lower priority. Across the life of the project there was substantial activity in the general area of authoring tools design and development. The central goal in all cases was to lower the costs of creating new scenarios, and most especially the costs of scripting simulated character behaviors. We explored various ideas for (a) structuring behavior specifications, and (b) enhancing reuse of resulting scenario and script elements, while (c) emphasizing reuse of existing tool assets. During the first cycle we developed rapid prototypes of key authoring capabilities using Microsoft Access for data and UI experiments. In the second cycle we exploited existing in-house code assets (e.g. SimVentive and GristEdit). During the third cycle we emphasized exploitation of common COTS tools, notably by developing a spreadsheet based format capable of encoding essentially all common classes of simulated agent behaviors (see Section 10.3).

10. Evaluate Successive Releases in Escalating Detail. Significant effort went into preparing for and carrying out system evaluation studies during this project. The original plan was to evaluate each of the three scheduled system releases. Introduction of additional human subjects review requirements led to delays in getting approvals for studies, and were largely responsible for changes in plans and delays in getting evaluation results to feed into further development. While we had planned for review of our formal evaluation studies by an outside Institutional Review Board (IRB) we had not planned on the need for full review of all studies by both our local IRB and the Army IRB as well.

Evaluation of the first Phase II prototype was carried out informally in-house. Project team members and other Stottler Henke employees were shown various pieces of the system and provided feedback, which was later organized to supplement prior requirements and designs. Major findings, as reported in the Year 1 Report, included identification of issues with the user interface, data representations, and authoring tools. The adoption of SimVentive as the environment for second cycle development was one outcome of this review.

We received approval from the Army IRB for the formative evaluation of the prototype from our second development cycle on 20 November 2006. The study involved bringing a variety of health care workers into our offices and observing their sessions with the prototype. The results of this study, included in our 11th project report, were broken down into training-related findings and system-interaction findings. There were three major training findings: (a) recognition that no simple scenario variants could bridge the large gaps between different medical specialties, and that dealing with differences in seniority/experience within one specialty would likely be challenge enough, (b) confirmation of our curriculum focus on diagnosis, treatment, and systemic response, including the appropriateness of asking students to engage in tasks (especially regarding response) that might normally be outside their range of responsibilities, and (c) advice on structuring an offering that might fit well into the Continuing Medical Education (CME) landscape. These finding shaped much of our work during the final development cycle,

including the emphasis on web delivery. The system interaction findings were generally more specific and were addressed by a plethora of specific changes in the system interface.

Our Final Evaluation study (based on the results of our third and final development cycle) was also somewhat delayed, leading to a final six-month project extension. To minimize IRB delays we were advised to forgo an observational study and use a survey approach. The results of this study are presented in Section 2.2.7.

11. Analyze Costs and Benefits. This task has been carried out as part of the preparation of this Final Report. The results are reported in Section 2.2.8.

12. Document Project Work. Throughout the life of this project we have carried out several different forms of project documentation. We have prepared Quarterly Reports, a First Year report, and now this Final Report. We participated in a project kickoff meeting at TATRC on 18 November 2004, briefing representatives from TATRC and OSD about our project plans, and delivered a First Year briefing 19 December 2005. In addition, we participated in the DOD Baseline Review of Medical Training in August 2005; due to a scheduling conflict Dr. Sowmya Ramachandran delivered the presentation for Dr. Domeshek. Most recently we participated in MMVR as an exhibitor in TATRC's trade-show booth on 30-31 January 2008. Finally, METTLE is discussed in a book chapter that is being prepared for an edited volume by Dr. Barbara Sorensen of AFRL.

8 Appendix B: Final System Architecture

8.1 The Enact Simulation Behavior Layer [Layer 7]

METTLE’s Enact layer is an implementation of the design first described in the Year 1 Report as the “METTLE Machine”—an abstraction capturing much of the structure of METTLE-style ITS scenarios. Enact is responsible for orchestrating the flow and interaction of *scenarios*, *scenes*, *actors*, *states*, and agent (including tutor) *behaviors* (all defined below). At its core, it supports a kind of **contextualized stimulus-response behavior modeling**, specifically including agent responses to student-initiative dialog cues. By relying on lower layers, the behaviors Enact manages can also include extended agent-initiative dialogs (using the Discuss layer) and student-model updates (using the GRAIN layer). METTLE introduces several specific kinds of behaviors related to patient management (e.g. running tests, delivering treatments, and updating the patient chart). Ultimately, all of these behaviors are managed through test and/or action extensions that plug into the Interpreter layer. Our Year 1 Report described and justified our notion of context-sensitive scripting. We include an updated version of that discussion in the following paragraphs.

METTLE is a scenario-based training system. That means limited simulation is carried out within the constraints of specific initial conditions, critical scheduled events, an acceptable envelope for student performance, and pre-identified principles and learning opportunities. Given METTLE’s emphasis on student interaction with simulated characters, we have been conceiving of the specification of METTLE scenarios as much like writing the script for a play—albeit a play that is potentially heavy on improvisation and light on dramatic arc. The implications of this approach are twofold: (1) a lot of scripting is required to prepare characters to play their roles (though with current technology pretty much any approach to naturalistic language-based interaction and coherent character presentation will require someone to author character-specific utterances), and (2) script elements will vary in how tightly bound they are to particular contexts (that is, many will be reusable, in different orders, in different scenes, across different scenarios).

Running with the “training scenario as scripted-play” metaphor, Enact clusters agent behavior specifications (script *lines*) into files (*scripts*). Given our scenario-based approach to training it should not be surprising that the major unit of scripting in Enact is the *scenario*. As with a standard play, we provide the option of breaking down a scenario into a sequence of *scenes* which can provide (a) a break in the action suitable for discussion and reflection, or (b) a chance to limit divergence in the simulation state of the world. To provide an additional level of flexibility in response to variations in the ways that scenarios and scenes can play out, Enact further allows each agent to have any number of internal *states* that can affect their response to events.

Scripts then are assigned and active for particular agents based on context—meaning a cascade of *actor*, *scenario*, *scene*, and *state*. For example, a simulated patient’s response to a question about whether they are feeling pain can be defined as holding only during a certain scene of a particular scenario, or even more narrowly to apply only in a state established after drugs have been administered and allowed to take effect.

Script files must be assigned to an actor, but can be assigned to that actor across all scenarios (thereby serving as default behaviors). Furthermore, script lines do not have to be completely specified in one place. The two main pieces of a line are the *cue* (or recognition conditions for when the agent should perform the behavior), and the *response* (what the agent should do when cued). Separate specification of cue and response means that commonly recurring aspects of behaviors (e.g. recognition that the student has asked a patient how they are feeling) can be separated from highly variable aspects of the same behaviors (e.g. what the patient says in response to being asked how they feel). Using this mechanism, the cues for most of the large set of patient lines can be stated once at the default level. The responses for

those lines can be defined so as to apply across an entire scenario, or during a particular scene of a scenario, or when the agent is in a particular state, or only given a combination of scene and state.

The above discussions accounts for the *contextualized* aspect of script lines. The cue/response structure of each script lines accounts for why we describe Enact behaviors as essentially *stimulus-response* pairs. Each agent script-line (behavior) is built from a cue (test) and response (action) where tests and actions are defined in terms of an extensible language framework provided by the Interpreter layer (Layer 6; see Section 8.4). As the prototypical example, Enact’s student-initiative dialog capability is supported by behaviors whose tests are based on the *hear* operator, and whose actions are based on the *say* operator. In its simplest form, such a script line might look like:

```
(-> _ (hear "How do you feel?") (say "I feel fine. "))10
```

In general, the problem of supporting student-led natural language dialogs with simulated agents is quite hard. We have chosen a relatively straightforward approach which we believe is appropriate to the application. METTLE uses variants on standard text-based Information Retrieval (IR) techniques to do what is often referred to as “bag-of-words” term-based matching between student inputs and pre-scripted triggers for script elements (again taking advantage of the context mechanisms described above).

Essentially, student text input is stripped of high-frequency low-content “stop” words (like “a”, “an”, “the”, etc.), and then the remaining words are stemmed down to canonical terms—that is root words created by removing standard prefixes and suffixes. The set of terms thus found in the student input is matched against the terms in pre-scripted triggers (which have been subject to the same pre-processing), subject to various weighting schemes. Similar techniques—sometimes involving additional processing like compressions of the term space such as that achieved by the singular value decomposition of lexical semantic analysis (Deerswester, et. al. 1990)—have been applied to language-based ITSs before (e.g. Graesser, Person, & Harter, 2001) but not, to our knowledge, in this exact kind of conversational context.

Obviously, this form of text interpretation is approximate and subject to well-known limitations. The “bag-of-words” approach throws away all information about the structure of an utterance so that the relationship of arguments to verbs and relationships among phrases and clauses is lost. To such a matcher “dog bites man” is the same as “man bites dog”. Similarly, without special support for tracking negation (which we have not yet implemented) “dog bites man” is much like “dog does not bite man”. Other confounds (depending on term weighting and the possible use of term-space compression techniques) include that “dog bites man” may be taken as more similar to “cat bites man” than “dog chomps man”.

Despite these limitations there are reasons to expect the technique may work reasonably well in the context where we are applying it, due to several aspects of the application:

- **Context (Expectations and Limitations):** The first and most pervasive effect that should make an IR approach workable is that we know a good bit about the kinds of things the student *should* be saying (expectations), and even more about what they *could* say that would be in any way interpretable by the system (limitations). Figure 13 present a picture of how to think about the spaces of things a simulated agent might hear from a students. We recognize three interestingly different kinds of utterances, and strive for different degrees of coverage in the set of “Expected Utterances”. Ideally the system should have *all* potentially appropriate on-task utterances among the set of things to which it is ready and able to respond. That is an exacting standard to meet, so we show the “Expected Utterances” box including most but not all of the “Appropriate On-Task Utterances” box. At the other extreme while there is a vast space of possible “Off Task Utterances” a student might indulge in, we make only a token effort to cover a small subset that have to do with establishing and maintaining social relationships—e.g. greetings, pleasantries,

¹⁰ Note that this common kind of line structure lends itself quite well to the spreadsheet-based encoding and editing described in 2.2.1.3. For many more examples of line structure, including substantially more complex rules, see Appendix B.

and such—which arguably are not really off task after all. The space of “Task Inappropriate Utterances” is perhaps the most interesting, because it is here that we find the keys to learning opportunities. Unfortunately, an ITS cannot be primed to recognize every kind of possible professional misstep at every moment in every context, so while it is critical that some of this space be included among the “Expected Utterances”, which subset is expected at any given time will have to be limited in keeping with the focus of the lessons of the scenario in play.



Figure 13. Venn Diagram of Student Utterance Spaces.

- **Assymetry:** To the extent that the various simulated characters are playing distinct roles—with different expectations about knowledge, responsibility, and authority—it is likely that there will be few utterances where references to any character can equally well fill any role in the sentence. An utterance like “Nurse, take the patient’s temperature” should never be confused with “Patient take the nurse’s temperature” (or any of a number of odder variants) because those variants would never make sense in the scenario and hence would never be contending in the space of possible matches. Upon reflection, even the “man bites dog” example is fairly contrived on this score.
- **Technical Vocabulary:** Synonymy can be a confounding factor as the “bite” “chomp” example suggests. Fortunately, medicine is a highly structured field with a vast technical vocabulary. Appropriate use of technical terms by professionals should often serve to lower the chances of the system missing a match due to synonymy. Furthermore, an elaborate technical vocabulary should increase the reliability of matches that hit on low-frequency technical terms. For instance, even in the midst of a scenario about anthrax, how many different expected utterances are likely to include the terms “pleural effusion”?

An important point is that using simple IR text matching significantly lowers the costs of authoring utterance expectations both in terms of expertise required (e.g. no need to understand complex parsing formalisms or semantic representation systems) and time taken. In order to end up with a practical maintainable system, it is an important goal of this project to manage authoring costs wherever we can. The earlier discussion of separately defining cues and responses, including large a large set of and pre-authored default cues for commonly recurring actions is an example of striving for such efficiency.

We note one final limitation of the system as it currently stands. All student utterances are directed towards a particular simulated character. There is currently no “overhearing” of (and possibly acting on) comments directed towards another character. We expect this limitation can be removed in the future.

A more complete survey of the test and action op-codes available in the Enact layer is provided in Appendix E. For now we simply note the general areas covered:

- **Scene Management:** The `scene` operator provides a means to move to a new scene of an ongoing scenario.
- **Agent State Management/Testing:** The `state` operator provides a means to test which state (or states) an agent is in, and to change those states under script-line control.
- **Agent Focus Management/Testing:** The `focus` operator provides a means to test which simulated agent currently has focus, and to change that focus under script-line control.

- **Dialog Management/Testing:** The `hear` operator allows an agent to test whether something in particular has been said to them. The `say` operator allows an agent to say something to the student. The `dialog` operator allows an agent to launch into an extended agent-initiative dialog structure, or to test whether such a dialog has already been executed.
- **Other Interaction Modes:** Enact has specialized operators defined for several other kinds of interaction including (a) tests for clicks on image hot regions, and choices on forms, and (b) actions for posting to the patient chart, or conveniently responding to patient examinations, test orders, or treatment requests.
- **Scenario Memory:** Enact provide the `line` operator to test whether certain behaviors have been triggered during a scenario run. Discuss provides a similar, but more specialized `node` operator for testing whether particular pieces of an extended dialog have been executed.

We note that one of the virtues of the original METTLE Machine concept, and its current realization in the Enact layer (built on the underlying Interpreter layer) is that it provides a clear way to manage changes and extensions to the set of meaningful tests and actions. It has succeeded in promote a new level of modularity in the design and implementation of METTLE and follow-on ITSs.

8.2 The Discuss Layer [Layer 6]

METTLE's Discuss layer is an implementation of the design for agent-initiative dialog control first described in the Year 1 Report. In contrast to the stimulus-response student-initiative dialog enabled by the Enact layer, this section focuses on (language-based) interactions in which the system—generally the tutor, but possibly some other simulated character temporarily playing a tutor-like role in the scenario—largely drives the shape and direction of the dialog. For reasons discussed in our proposal, we expect such discussions will often to take the form of Socratic dialogs exploring the rationale underlying student actions (or potentially the student's interpretations of others' actions).

Again, as noted in our proposal, Socratic tutoring dialogs are a long-standing goal of many in the ITS research community; further, Stottler Henke, in a project under Dr. Domeshek's leadership, developed an initial prototype capability in this area. As described in our Year 1 report, as part of the METTLE project we identified a set of module requirements and developed a design intended to refine our earlier work. Major goals included developing a dialog module that was (a) more robust, (b) reusable, (c) simpler to use, but (d) flexible, and (e) extensible. The implementation described here, in large part based on its use of the underlying Interpreter layer, meets these criteria.

In our Year 1 report we identified the following requirements for a proposed discussion module (each set of requirement bullet points is followed by our assessment, set off in italics, of how the final implementation addresses those points):

- **Reusable Dialog Package:** The goal was to build a general reusable package that can be applied across a range of current and future projects that need an ability to script and control Socratic dialogs.
 - The package should have as few required dependencies as possible, and those it does have should be as generically acceptable as possible. In particular that means we should avoid building GRIST in as a necessary part of the system.
 - Where possible, connections to other packages should be factored out and wrapped in such a way that it is possible to plug in alternate means of accomplishing desired ends (e.g. representation, matching, authoring, dialog output, and dialog input).
 - ✓ *This goal and both of its sub-goals are well satisfied by the current implementation. The extensible Interpreter layer is a tremendous boon here. Built-in tests and actions for the tree-structured dialogs are very simple, primarily reusing such operators as `hear`, `choose`, and `say`, and paralleling the Enact `line` operator with a similar `node` operator to provide*

memory for which parts of the dialog have been executed. Supplementing this basic set of operators is easily done by extending the underlying interpreter configuration.

- **Socratic Dialog:** By Socratic dialog we mean a primarily question/answer dialog, largely under tutor control, with the tutor mostly asking the questions and the student mostly answering them.
 - Tutor dialog control should make construction of this package easier in that (i) so long as the tutor sets the agenda, the topics of conversation can be limited, and (ii) so long as the tutor asks the questions, the interpretation of student input can be done in a largely known context.
 - ✓ *The design as sketched in the Year 1 Report took advantage of these features of the problems, and the implementation is faithful to those aspects of the design.*
- **Student Initiative:** Some limited amount of student initiative should be allowed in the resulting dialogs. Possible examples include (a) requesting clarification of tutor questions, (b) asking a limited range of questions about facts of the domain or scenario, (c) making comments on their state of understanding or feeling, (d) making requests relative to the control of the discussion, including referring back to previous topics or parts of the discussion.
 - Each kind of student initiative interaction we want to support likely requires its own extra data structures and control regimen. The design should allow for their addition in a nicely compositional/extensible way.
 - ✓ *The Discuss layer provides infrastructure for addressing at least items a, b, and c, plus limited cases of d. The primary response was to introduce a marker that can be assigned to possible student responses to mark them as digressions. This provides a flag that can allow the discussion machinery to deal with such inputs locally without otherwise affecting the overall flow of the larger dialog. In addition, hierarchical dialog contexts allow responses expected by still-active dialog nodes to be recognized (and to affect dialog flow) even when other nodes intervene.*
- **Student Input:** Student input to the dialog should be as natural as possible within the bounds of (a) what media formats are supportable by a particular platform, and (b) requirements for authoring effort and interpretation accuracy. If a system is bothering to use Socratic discussion at all, it probably defeats the purpose to lean too heavily on strong user input constraint (e.g. small multiple-choice).
 - Natural language input is the gold standard, with speech input processing the ideal. The ability to coordinate language with gestures indicating items in the (simulated) environment is also highly desirable.
 - We don't expect to meet the ideal in all circumstances. We need a pluggable range of input mechanisms that include more limited language (phrases and blank-filling) and gesture use, along with forms, pick-lists, "utterance constructors" and other expressive input mechanisms.
 - ✓ *The Discuss layer meets these requirements. It inherently supports the same style of natural language input processing as Enact (bag-of-words matching against expectations) as well as several styles of multiple-choice (options enumerated within the dialog tree, either locally or inherited, and in either fixed or randomized orders, or options defined as subsets of concept spaces and displayed as hierarchical trees). Experiments with COTS speech recognition technology indicate that speech-based input is highly feasible. Again the extensible Interpreter layer makes it easy to plug in other kinds of input recognition technology, or to add coordinated testing of other environmental state (including student gestures).*
- **Tutor Output:** Tutor output contributions to the dialog should be as natural and rich as possible, again within the bounds of (a) what media forms are supportable by a particular platform, and (b) requirements for authoring effort and interpretation accuracy.

- HTML or other multimedia output may require access to (sets of) files produced by other (COTS) authoring tools. This package cannot assume that tutor output is necessarily just a simple text string authored using a type-in box.
- An acceptable level of text generation is probably easier than accurate language understanding, but appropriately expressive speech output is still quite hard.
- ✓ *The Discuss layer is capable of meeting these requirements, though in the current application it has proven sufficient to limit output to plain text and hand-authored HTML. Should it prove useful to separate output authoring from dialog scripting, a simple Interpreter extension would suffice (i.e. a `file` operator that outputs the contents of an externally prepared HTML file). Likewise, though we have not found it necessary to generate templated text, a variant of the basic `say` operation that can substitute in alternate values from the environment is a logical interpreter extension.*
- **Package Integration:** The dialog capabilities should be integratable with a range of tutoring systems that deal with a range of other kinds of input and output.
 - Socratic dialog control will have to be accessible to the embedding tutoring system through some kind of API.
 - Accepting student input (including text and forms, etc.) may have to be routed to some input-handler/display-mechanism established by the embedding tutoring system.
 - Accepting gestural input requires some kind of hooks into the larger (simulation) environment.
 - Tutor utterances (including media) may have to be routed to some output-handler/display-mechanism established by the embedding tutoring system.
- ✓ *The Interpreter layer provides our way to access and affect any larger environment in which a discussion is embedded.*
- **Authoring Tools:** The dialog package should come with an easy-to-use authoring tool—ideally one amenable to end-user (SME) dialog authoring.
 - Keep the authoring GUI design nicely factored from the core data-structures and algorithms.
 - *As illustrated in the Year 1 Report a Discussion Editor was developed early in the project. Unfortunately, that editor was not ported over to the newer Discussion layer implementation. This was largely because the editor was judged to suffer from some of the usability problems that motivated our exploration of spreadsheet-based authoring support; however spreadsheet-based authoring of dialog trees was deemed inappropriate due to their highly nested structure. Discussion authoring for the final version of the system was carried out using a COTS text editor to create files as illustrated in Appendix F. This robust editor, especially with its built-in support for parentheses-balancing and automated indentation, proved quite adequate to the task for an author familiar with the dialog tree structure and the available interpreter operators.*
- **Demo and Test Constraints:** Ideally, the dialog capabilities should make it quick and easy to (a) load in new dialog scripts, (b) jump to particular places in a loaded dialog script, (c) allow for automation of the student's part in the dialog, and (d) enable undo/redo capabilities to explore alternate dialog paths.
 - Avoid excessive heavy-weight components that take a long time to load or reset.
 - This is a difficult requirement to meet in full, particularly because it does not suffice to have an undo facility in the discussion runtime; in addition, all external systems used for world modeling and interaction must support undo as well!

- *We did not address this requirement, other than to keep the entire dialog machinery quite light weight—dialogs load and run very quickly. We did not attempt to introduce an “undo” capability into the Interpreter layer, judging that to be far too much complexity and overhead to impose on what is otherwise also a relatively simple light-weight framework.*

Our Year 1 Report also included a high-level design description of dialog module intended to meet the requirements listed above. What follows is a slightly simplified version of that design reflecting the ultimate implementation. As a matter of conceptualization/vocabulary, we think of the system's Socratic **Discussions** as being organized around **Topics** each of which organizes a collection of **Points**, which in turn can be thought of as sets of possible **Queries** that can elicit student **Responses**. In addition, specification of conditional control and side-effects for all these discussion-pieces (discussion tree nodes) is based on Interpreter layer tests and actions.

- **Discussion** nodes are the roots of tree structures that represent scripts for a Socratic dialog; we allow for wide variation in complexity. Many fancy dialog structures are possible, but simple structures remain reasonably simple to script.
- **Topic** nodes have trigger conditions that determine when/whether they get introduced into the Discussion.
- **Point** nodes can have entry and exit conditions and can be organized as (ordered) hierarchical arguments (which potentially includes enumerated lists). The basic idea is that a Point tree structures the discussion of a Topic as an attempt to get the student to see/acknowledge some key facts/insights. The tree structure represents arguments that support higher-level Points, and is explored at variable depth, as required by a student's individual history of seeing/accepting those Points.
- **Query** nodes are specifications (optionally with entry conditions) for generating tutor output and accepting user input; they offer alternate ways to explore a Point.
- **Response** nodes enumerate the available interpretations for student Query inputs; they define the match criteria for each interpretation, and optionally can specify side-effecting consequences.

The intent is that the several grain-sizes of Discussion, Topic, Point, Query, and Response will be clear and distinct enough that authors won't get hung up wondering which one(s) to use in any particular situation. On this score, the most likely confusion is probably going to be about when something is a Topic versus when it is a Point. In what follows we work out some of the details bearing on these structures which should clarify their use:

1. The system should be able to introduce new Topics based on conditions such as (a) the student's performance in a coupled simulation, (b) the student's performance in earlier parts of the Discussion, and (c) aspects of the system's long-term student model. Any such conditions will be checked using a set of Tests.
2. For convenience and clarity, Topics can be nested under other Topics, which means that a nested Topic is only considered for activation when its parent Topic has been activated. Put another way, the trigger conditions on a Topic implicitly include the trigger Tests from all ancestor Topics in the packaging hierarchy; also a nested Topic is implicitly constrained to be executed *after* its ancestor Topics.
3. Activation of a Topic leads to what is essentially a preorder traversal of the Topic's Point tree. If a Point has a test condition, it is checked at the time the traversal reaches the Point; the Point will immediately succeed if its test is true (that is, if the student has already said/acknowledged things that are logically equivalent to the target of the Point).
4. If a Point becomes active (traversal reaches it, and its test is not yet satisfied) then the Queries for that Point are attempted, in order, until either the Point's test is satisfied, or we run out of Queries. If a Point has nested Points (i.e. it is prepared to descend to a more detailed level of argument to

get the Student to discover/accept the original Point) then its Query list can contain a special flag which when reached means the discussion engine should start traversing the nested Points list.

5. Each Query can also have a test that is checked before the Query is attempted. If the test passes, then (a) the Query action is performed (generally asking the student a question), (b) potentially relevant Responses are identified and queued up, and (c) the discussion engine stops and waits to be re-invoked by the controlling program (which will generally happen after some additional student input has been received).
6. Following a student input and re-invocation of the discussion engine, the set of active Responses is scanned to see which ones have been satisfied. Each such Response is processed, which means (a) any action it carries is executed, (b) changes in Topic activation are noted, and (c) the current Point stack is scanned to identify any active Points that may have become satisfied (checking higher-level points first). Finally, if no higher-level point has been satisfied, then (d) the Response has the option of launching into its own nested Point tree (for targeted follow-ups to particular student inputs).
7. A Point node is closed out when (a) its test is matched [success], (b) the test of an ancestor point is matched [neutral], or (c) the algorithm has scanned past the Point's last Query [failure]. When a node is closed out, an optional success or failure action can be executed. Points should be tied to the tutoring environment's curriculum model, and success/failure used to update the student model.
8. A Query can specify a set of Responses, and/or can depend on the Responses defined by the stack of Point, Topic, and Discussion above it (a flag controls whether to include inherited Responses or not). Alternately, a Response can reference a (part of a) concept space which implicitly defines a set of Responses (e.g. a pre-defined hierarchy of medical findings). A Response test generally includes (some) hear forms. Free text Queries use bag-of-words based matching across the set of active Responses' hear texts (with confirmation of the highest ranking matches among which Students can choose their intended meaning). Queries flagged as multiple-choice present such hear forms as options (in fixed or randomized order); if a single Response has multiple hear forms then first of them is used to generate its multiple-choice option.
9. In keeping with the notion that we might want to apply this system without getting into a pervasive and/or heavyweight world-model, the discussion engine maintains a set of simple (Boolean) flags for every Topic, Point, Query, and Response. These flags get automatically set when the node is attempted (or in the case of Points, separate flags are set to note activation versus satisfaction). Without any other representation, this enable authors to set up exit-tests on Points as simple Boolean combinations of the flags associated with key Responses nested under those Points.

8.3 The GRAIN Layer [Layer 5]

During work on the earlier ComMentor system, we first developed the idea of defining a layer of ITS implementation that would capture generic representations for *courses* and *curricula*, *instruction* and *exercises*, *students* and the accumulating *records* of their experience with the system. We envisioned a suite of tools that we dubbed the General Runtime and Authoring for INstruction (GRAIN). In many ways, the role envisioned for GRAIN is similar to that of a more generic Learning Management System (LMS), but slightly specialized to exploit a more structured notion of curriculum and associated student modeling, so as to enable adaptive delivery of instruction and exercises.

In the course of METTLE development we made our first attempt to implement this idea. The GRAIN layer pictured in Figure 2 represents that implementation and its place in the overall METTLE system. This layer provides data structures and persistence handling for the six major classes of objects listed above, plus one addition. We describe each of those structures here:

- **Role:** The single new concept introduced in the METTLE implementation of GRAIN is the notion of a *Role*. Roles are intended to represent major job/training categories such as (in the medical world) doctor, nurse, or administrator. A Role is then associated with a set of training needs, as represented by (variants of) Courses and associated Curriculum points.
- **Course:** A Course is intended to represent a collection of Instruction and Exercise elements that together address a set of Curriculum points in ways appropriate to some collection of Roles. In the context of METTLE, we thought of ourselves as developing materials for a course in “CBR Hospital Response.”
- **Curriculum Point:** A Curriculum Point is a single node in a hierarchically organized collection of such nodes where each is intended to represent some knowledge and/or skill that can be taught (to students in some Roles) using some Instruction and/or Exercise items. METTLE, for instance, defines a set of Curriculum points related to emergency department diagnosis, treatment, and activity sequencing.
- **Instruction:** An Instructional item is essentially a URL (a reference to a web-deliverable piece of rich media) that is labeled with some additional metadata, such as the Curriculum point it addresses, the roles it is appropriate to, the mode in which it addresses it (e.g. whether it serves as an *introduction*, an *example*, a *remediation*, etc.; a set of categories roughly aligned with the “cognitive media types” identified by Recker, et al. (1995)), and sequencing information. In the end, we did not develop this concept in any depth for METTLE, and we think of it largely as a placeholder for future development.
- **Exercise:** An Exercise item is similar to an Instructional item, but references a directory on the server and assumes that directory contains an exercise definition file along with any other the server needs to drive an extended interactive experience. The exercise definition file is the point where generic GRAIN structures bridge to the more specific Enact ITS structures (in theory there could be different kinds of Exercises available as part of a single Course, but METTLE only uses Enact). Like Instruction items, Exercises should also carry metadata characterizing dimensions such as the Curriculum points they address, the roles they are appropriate to, and the expected mode of use (e.g. practice versus assessment). Again, METTLE did not develop this aspect of GRAIN, as there were not enough Exercises to make metadata-driven exercise selection or sequencing worth pursuing.
- **Student:** A student is simply a user known to a GRAIN installation (assigned a set of Roles) who can log in and access relevant Instruction and Exercises. The most important aspect of having a Student representation is that it allows the accumulation of Student-specific Records of exposure to Instruction and performance during Exercises. In a more fully developed version of GRAIN, Students would no doubt accumulate additional metadata.
- **Record:** A student Record element is a time-stamped indicator that something of interest happened in the Student’s use of the GRAIN system. The system keeps Records associated with Instructional items (noting when a Student views them), Exercises (noting when a Student starts and finishes an Exercise), and Curriculum points (noting when they succeed or fail in application of the point during an Exercise). It is up to each specific Exercise interpreter to determine what it considers success or failure. For instance, the Enact interpreter considers Student requests for help, Tutor decisions to prompt the Student, or inappropriate behaviors as failures, while correct behaviors are counted as successes.

The GRAIN layer plays a useful role in the METTLE implementation, but it is clearly a preliminary implementation, and one that may reasonably be swapped out should METTLE be delivered in the context of a pre-existing LMS.

8.4 The Interpreter Layer [Layer 4]

The Interpreter layer is a core technology that, as has been suggested at several points above, enables and unifies many of the capabilities of the overall system. The Interpreter is an extensible object-oriented framework for defining and composing small instruction languages that enable simple expressions to be evaluated, either to produce true/false values (tests) or to produce effects within the system. The interpreter core provides a simple API and a set of classes that implement that API for a basic set of logic and control operators. Each new operator is built as a class that implements the same API. Instances of the core logic/control operator classes plus any number of custom operator classes can be registered with a top-level interpreter instance.

The main methods of the API are `validate()`, `test()`, and `act()`, plus a variant of `test()` called `rate()` that can be used when the result of a test is usefully thought of as a graded value (e.g. a real number from 0 to 1) rather than a Boolean value (true or false).

- **Validate():** Implementations of `validate` simply check that the given form is a valid expression using the class's operator. Typical actions performed include checking for the right number and type of arguments. In the case of composable operators (e.g. those that allow recursive structure) the `validate` check dispatches to check on sub-forms as needed. This method is very useful in support of authoring, to try and ensure that test and action expressions included in script lines or dialogs are well-formed.
- **Test():** Implementations of `test` evaluate a form and return a Boolean value of true or false. Example tests include `hear` (testing what was said in the most recent Student utterance), `line` (testing whether a specified script line has been run), and `node` (testing whether a specified node in a dialog tree has been interpreted).
- **Rate():** Implementations of `rate` evaluate a form and return a real value from 0 to 1 indicating the strength of a test result. The primary example of this method being implemented in an interesting way is for the `hear` operator; in that case, the actual text typed by the Student is compared against the text argument to the `hear` expression using bag-of-words comparison, and a weighted score is returned indicating how strong a match was found.
- **Act():** Implementations of `act` evaluate a form strictly for its side-effects; they return no value. Example acts include `say` (causing a character to say something to the Student) and `post` (causing information to be added to a designated part of a patient chart).

There are five core logical operators provided to support composition of *tests*:

- **and:** The `and` operator class provides an implementation of `test` that combines the logical values of its arguments such that it only returns true if *all* of its arguments evaluate to true. It implements what is commonly referred to as “short circuit” semantics, meaning it stops checking its arguments as soon as it finds one that evaluates to false. Its implementation of `rate` combines the ratings of its sub-forms by returning the *minimum* value.
- **or:** The `or` operator class provides an implementation of `test` that combines the logical values of its arguments such that it returns true if *any* of its arguments evaluate to true. It implements “short circuit” semantics, here meaning it stops checking its arguments as soon as it finds one that evaluates to true. Its implementation of `rate` returns the *maximum* of its sub-form values.
- **not:** The `not` operator class provides an implementation of `test` that takes a single argument and returns the negations of whatever value that sub-form evaluates to. Its implementation of `rate` returns one minus the rating of its sub-form.
- **true:** The `true` operator class provides an implementation of `test` that takes no arguments and always returns a Boolean value of true. It is primarily useful for forcing a test value when experimenting during development. Its implementation of `rate` always returns the value 1.0.

- **false:** The `false` operator class provides an implementation of `test` that takes no arguments and always returns a Boolean value of false. It is primarily useful for forcing a desired test value, as when experimenting with rules during development. Its implementation of `rate` always returns the value 0.

There are four core control operators provided to support composition of *actions*:

- **seq:** The `seq` operator class provides an implementation of `act` that combines the actions of its argument sub-forms so that each sub-form is executed in sequence.
- **any:** The `any` operator class provides an implementation of `act` that chooses one of its argument sub-forms at random to execute each time it is evaluated.
- **alt:** The `alt` operator class provides an implementation of `act` that each time it is evaluated chooses the next one of its argument sub-forms to execute; once it has been evaluated as many times as it has arguments, it continues to re-execute its final argument on each subsequent evaluation.
- **nop:** The `nop` operator class provides an implementation of `act` that does nothing when it is evaluated. It is primarily useful as a final sub-form in an `alt` form (when you don't want a final action endlessly repeated on subsequent evaluations).

The Interpreter layer does provide some additional, more specialized capabilities in its API. One such method, `collect()` makes it possible to find all forms with some particular operator, even though they may be nested arbitrarily deeply inside other forms (e.g. inside the logical and control operators just described). This `collect()` method is useful, for instance, when you want to find all the `hear` forms when trying to build up a textual index. Another set of such methods (`log!`, `clear!`, `current?`, and `history?`) allows each interpreter to optionally maintain its own memory, for example to track forms it has evaluated. These methods are used to implement the `line` and `node` memory operators.

8.5 The Concept Layer [Layer 3]

The Concept layer provides a very simple hierarchically structured data store with slot/filler capabilities. It organizes its concept hierarchies into disjoint spaces and allows for linkages within and across spaces. Some of the major concept spaces used by METTLE include `conditions`, `findings`, `tests`, and `treatments`. The most common sort of slot/fillers are URLs that provide access to external reference materials about particular concepts (e.g. CDC web pages, or military reference PDF file extracts describing particular conditions or treatments). By virtue of being built on top of the Registry layer (see 2.2.2), it proved relatively easy to build a simple concept space editor.

Concept spaces (and sub-spaces) are commonly used to structure sets of choices for users in METTLE. This is done in two different ways. The more elaborate approach allows rapid definition of customized HTML form displays, such as those METTLE provides for ordering tests and treatments. The simpler approach automatically generates checkbox trees derived from (parts of) these concept hierarchies and presents them directly as multiple-choice Student response options during dialogs.

An overdependence on conceptual pattern matching and instantiation was identified as one of the cost-drivers and limitations of our earlier ComMentor ITS. In METTLE, we generally strove to minimize reliance on such relatively complex techniques, which led to the much more restricted role for conceptual structures just sketched. Nonetheless, higher end representational capabilities have their place if used judiciously. Thus, in future development, one obvious direction to pursue is to carefully strengthen concept structure matching and instantiation capabilities, while surfacing those enhanced capabilities as operators in the Interpreter layer. This approach would ensure smooth integration of conceptual structure tests and manipulations as but one capability among METTLE's wider suite of lighter-weight test and action operators.

9 Appendix C: Sample METTLE Scenario Interaction

Each scenario starts with a briefing. What follows is the briefing for our anthrax diagnosis scenario:

Scenario Briefing for "Mystery Patient 1"

Play Scenario

Help

Quit

Role and Setting

In this scenario you are an Emergency Department (ED) doctor at **City Hospital** in a mid-sized northeastern city. It is winter. The ED is quite busy, in part because of the recent statewide influenza outbreak.

Patient

Mr. Smith is a 48 year old man who entered the ED on February 12 at 7 PM. He is brought in by his family who are quite concerned. He is usually well and his only history is mild hypertension, now controlled on Hydrochlorothiazide 12.5 mg per day. He was seen in this same ED yesterday a little over 24 hours ago. At that time he complained of:

Fever of 100.8 orally.

Fatigue and muscle aches all over.

Dry cough.

Headache.

Influenza A is present in the community. He was diagnosed with Influenza and begun on "Tamiflu" (Oseltamivir) which he has been taking at home. No lab work was done yesterday. A rapid influenza test is available, but per hospital guidelines patients with typical influenza symptoms are not having this test done because of a relative shortage of the test kits.

The family notes that he went to a basketball game three to four days before falling ill, and that his cousin who went with him is also now home with the flu.

Current Complaint

In the past 24 hours his condition has worsened in several ways:

His fever is now 102.2 orally and he is having sweats.

He now has chest discomfort on deep breath.

He is short of breath at rest.

Headache is worse.

Weakness is worse and the patient now feels unable to get out of bed.

His family feels that at times in the last few hours he seems slightly confused and uninterested in his surroundings.

Vitals

Time	Temp	Heart Rate	Resp. Rate	BP	O2 Sat.	Weight
02/12/07-19:08	102.2 (tympanic)	118	28/min	115/72	93% (room air)	195 lbs

Physical Exam

Constitutional	Patient is slightly overweight. He looks ill and does not greet the examiner. The patient lies in bed looking at the ceiling or keeps his eyes closed.
Skin	Normal
HEENT	The head is normocephalic and atraumatic. The neck seems slightly stiff and the patient has pain on movement of the neck.
Chest	The chest shows slight rales diffusely.
Heart	Regularly irregular. Heart rate slightly rapid. There is no murmur. Heart sounds are normal.
Abdomen	Normal exam.
Extremities	Extremities are slightly cool. There is very slight cyanosis of the nail beds.
Neurological	Patient is well oriented when pressed and there is no focal neurologic abnormality. However the patient at times seems uninterested and does not always answer questions. He has neck stiffness as noted above.

Instructions

When you are ready, please click the "Play Scenario" button to move to the main scenario player screen. The findings described in this introductory briefing will also be available in the scenario as part of the patient chart.

When the student clicks the "Play Scenario" button they are taken to the Scenario Player screen shown below in Figure 14. As is the case for all the screen shots to follow, this is an image of a pop-up web browser window filled with styled HTML text, buttons, fields, images, tables, and layout regions.

A reasonable place for the student to start is with the basics of a patient interview. Here we show the student having typed a typical query intended to elicit the patient's chief complaint. In response to a click on the "Try" button, the system has offered a range of possible interpretations for what the student meant to say. First, note that the system can tolerate a reasonable degree of misspelling in student input: "Wht seems two bee your problm tuday?" is correctly interpreted. Second, note that what the student says and what the system expects to hear need not be an exact match in content or syntax. Finally, note that the system prompts the user for confirmation of its best-guess interpretation. The top-ranked match is checked by default.

Figure 15 shows the student question from Figure 14 echoed near the top of the left side of the screen, followed by the simulated patient's answer; in addition, the system can play a sound file so the student gets to hear the patient answer the question. Figure 15 also shows the student asking a follow-up question. The student is free to check off multiple meanings if their input turns out to be equivalent to more than one expected meaning. Note also that we have experimented with using commercial off-the-shelf speech recognition software, such as Dragon "*Naturally Speaking*" to interpret student input—using it to automatically type what the student says into the "Try" text box—and it has proven to work quite well.

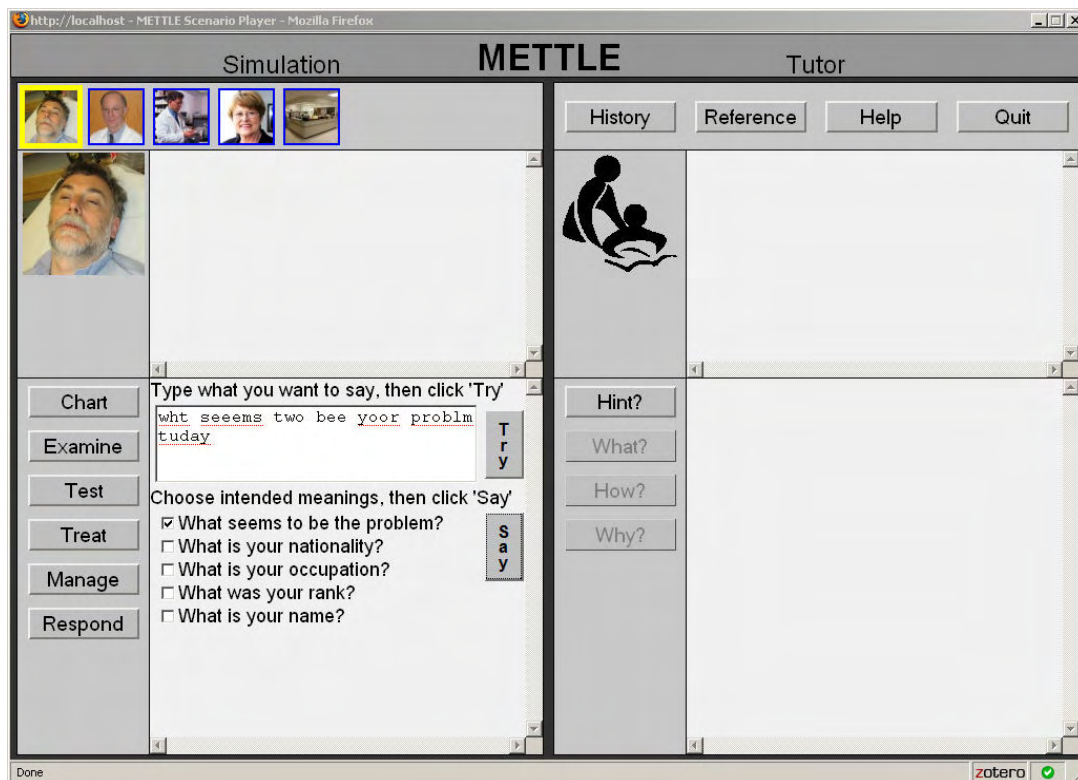


Figure 14. A Reasonable First Patient Interview Question, Badly Misspelled.

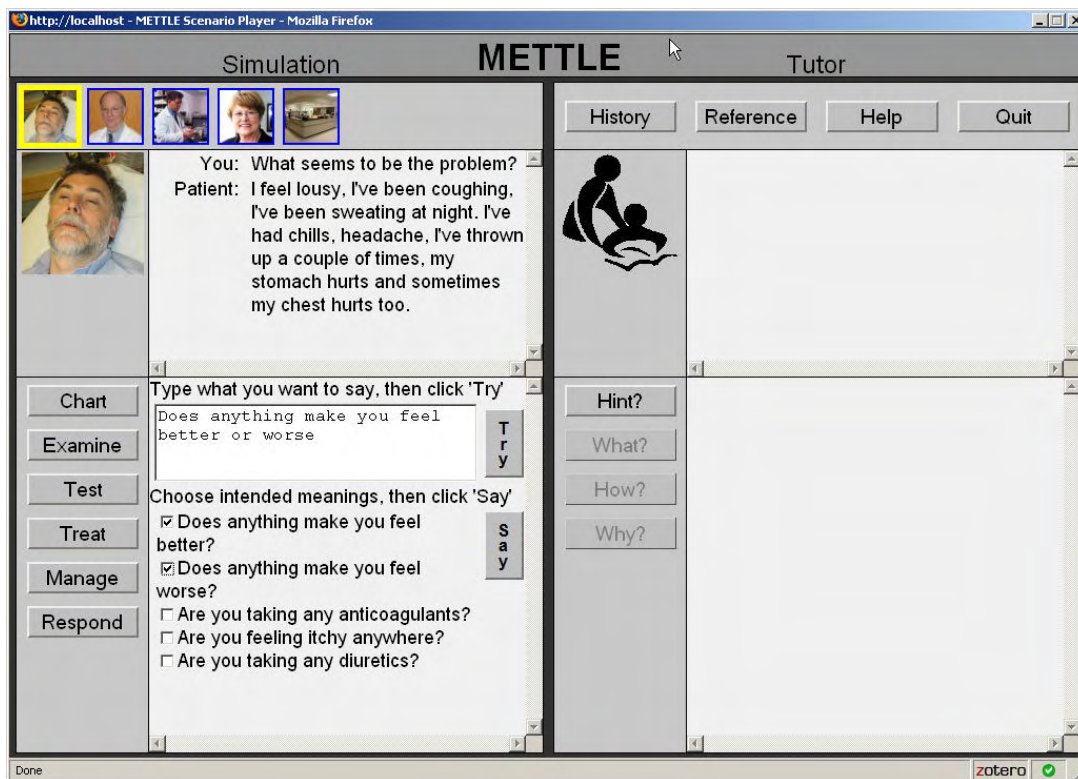


Figure 15. A Follow-up Patient Interview Question with Multiple Meanings.

In addition to conducting a patient interview, METTLE allows the student to conduct a simulated patient examination. Clicking the 'Examine' button in Figure 15 pops open the Patient Examination window shown in Figure 16. Front and back full-body views of the patient provide clickable hot regions for inspectable parts of the body. Figure 16 shows examination results after clicking on the patient's eyes, and Figure 17 shows results of clicking on the nose. The system's response includes a detailed image on the right-hand side of the window, and summarized findings at the bottom of the window.

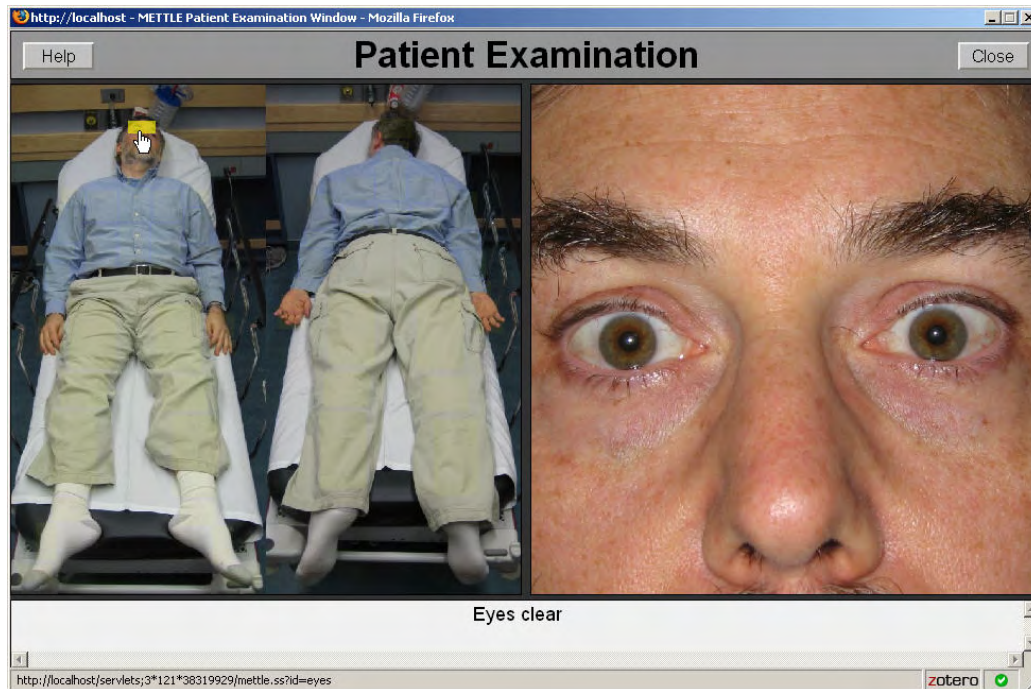


Figure 16. Simulated Patient Examination – The Eyes.

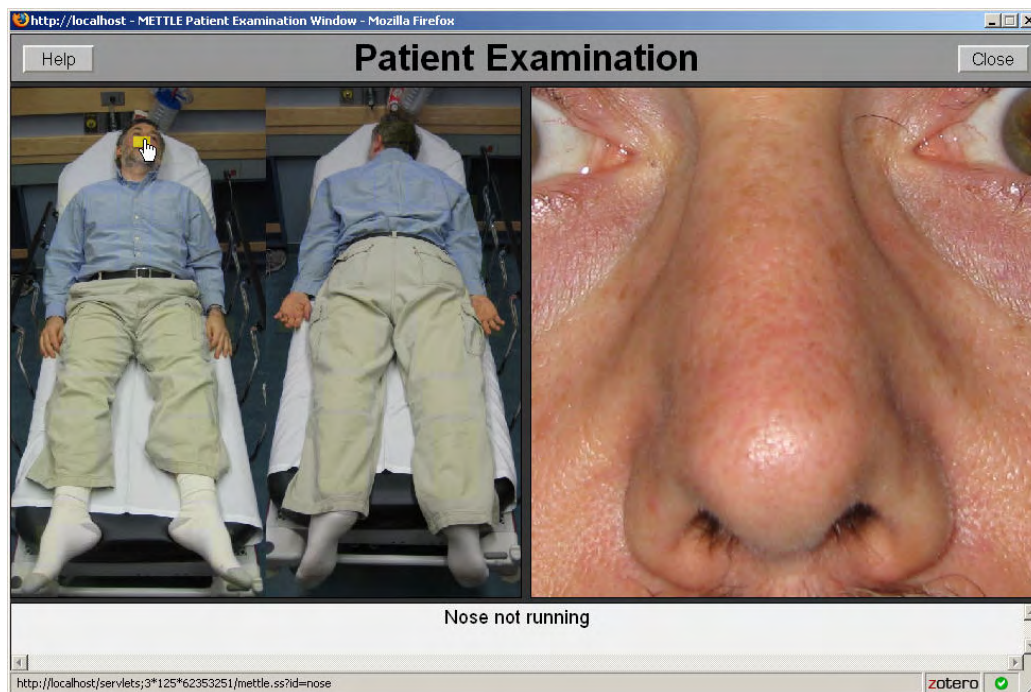


Figure 17. Simulated Patient Examination – The Nose.

Back in the main Scenario Player window, the student may decide to ask for guidance from the Tutor (over on the right side of the screen). As shown in Figure 18, clicking the ‘Hint?’ button is equivalent to asking the Tutor “Please give me a hint.” At this early stage of the scenario, the Tutor suggests taking care of basic Emergency Department (ED) procedure. In Figure 19, a click on the ‘What?’ button—a request for more specific direction from the Tutor—results in a suggestion for a specific action. Figure 20 shows the result of clicking the ‘How?’ button, and Figure 21 shows the Tutor’s response to the ‘Why?’ button. Note that the button for each successively more detailed piece of advice only becomes active after the earlier, more general buttons have been clicked. Also note that if the Tutor has several lines of advice active, then clicking multiple times on the ‘Hint’ button cycles through those topics. In this scenario run, since we have already done some basic interviewing and examination, those Tutor advice topics are no longer active.



Figure 18. Asking for a *Hint*.

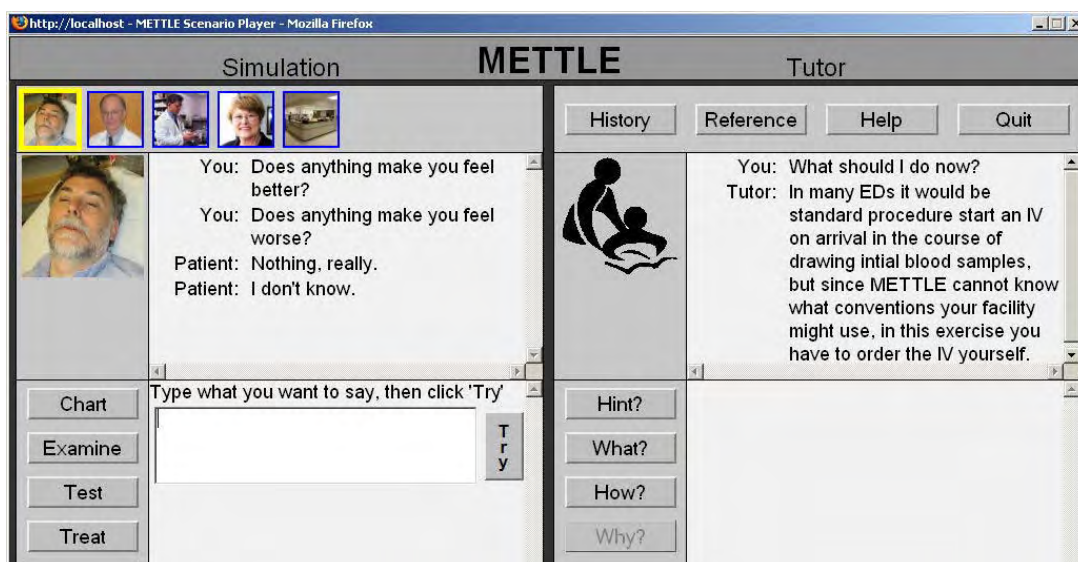


Figure 19. Asking for More Explicit Direction on *What* to Do.



Figure 20. Asking for Explanation of *How* to Take the Recommended Action.



Figure 21. Asking for an Explanation of *Why* to do the Recommended Action.

A student who decided to take the Tutor's advice would click the 'Treat' button and get a pop-up menu of general treatment categories as shown in Figure 22. In this case, selecting 'Fluids' brings up the order-form window shown in Figure 23.

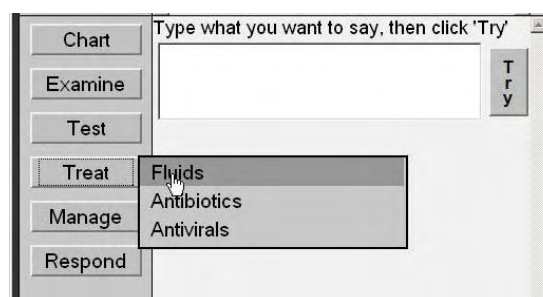
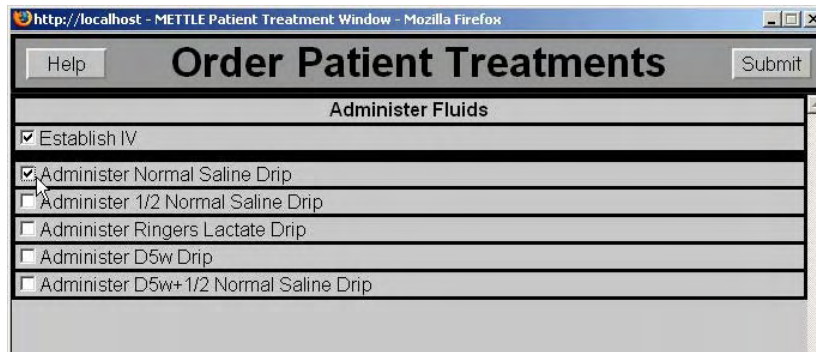


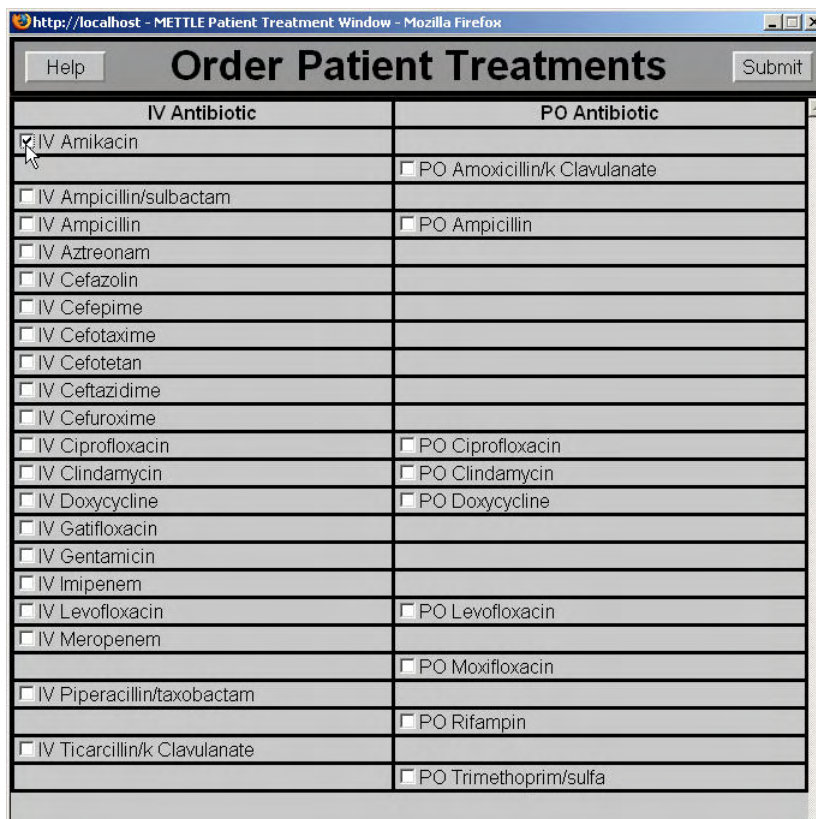
Figure 22. Pop-Up Menu of Treatment Categories.



Administer Fluids	
<input checked="" type="checkbox"/>	Establish IV
<input checked="" type="checkbox"/>	Administer Normal Saline Drip
<input type="checkbox"/>	Administer 1/2 Normal Saline Drip
<input type="checkbox"/>	Administer Ringers Lactate Drip
<input type="checkbox"/>	Administer D5w Drip
<input type="checkbox"/>	Administer D5w+ 1/2 Normal Saline Drip

Figure 23. Patient Treatment Order Form with “Administer Fluids” Options.

As suggested by figure Figure 22, other categories of treatment are available. Figure 24 shows the treatment order-form under the ‘Antibiotics’ item in the ‘Treat’ menu. In this sequence, we assume that, in response to the patient’s severe illness, the student has decided to put the patient IV Amikacin. .Figure 25 then shows the Tutor’s reaction to this action. In general the Tutor may decided to give immediate positive or negative feedback when it observes the student performing what are judged to be contextually appropriate or inappropriate actions.



IV Antibiotic	PO Antibiotic
<input checked="" type="checkbox"/> IV Amikacin	<input type="checkbox"/> PO Amoxicillin/k Clavulanate
<input type="checkbox"/> IV Ampicillin/sulbactam	<input type="checkbox"/> PO Ampicillin
<input type="checkbox"/> IV Ampicillin	<input type="checkbox"/> PO Ciprofloxacin
<input type="checkbox"/> IV Aztreonam	<input type="checkbox"/> PO Clindamycin
<input type="checkbox"/> IV Cefazolin	<input type="checkbox"/> PO Doxycycline
<input type="checkbox"/> IV Cefepime	<input type="checkbox"/> PO Levofloxacin
<input type="checkbox"/> IV Cefotaxime	<input type="checkbox"/> PO Moxifloxacin
<input type="checkbox"/> IV Cefotetan	<input type="checkbox"/> PO Rifampin
<input type="checkbox"/> IV Ceftazidime	<input type="checkbox"/> PO Trimethoprim/sulfa
<input type="checkbox"/> IV Cefuroxime	
<input type="checkbox"/> IV Ciprofloxacin	
<input type="checkbox"/> IV Clindamycin	
<input type="checkbox"/> IV Doxycycline	
<input type="checkbox"/> IV Gatifloxacin	
<input type="checkbox"/> IV Gentamicin	
<input type="checkbox"/> IV Imipenem	
<input type="checkbox"/> IV Levofloxacin	
<input type="checkbox"/> IV Meropenem	
<input type="checkbox"/> IV Piperacillin/tazobactam	
<input type="checkbox"/> IV Ticarcillin/k Clavulanate	

Figure 24. Patient Treatment Order Form with “Administer Antibiotics” Options.

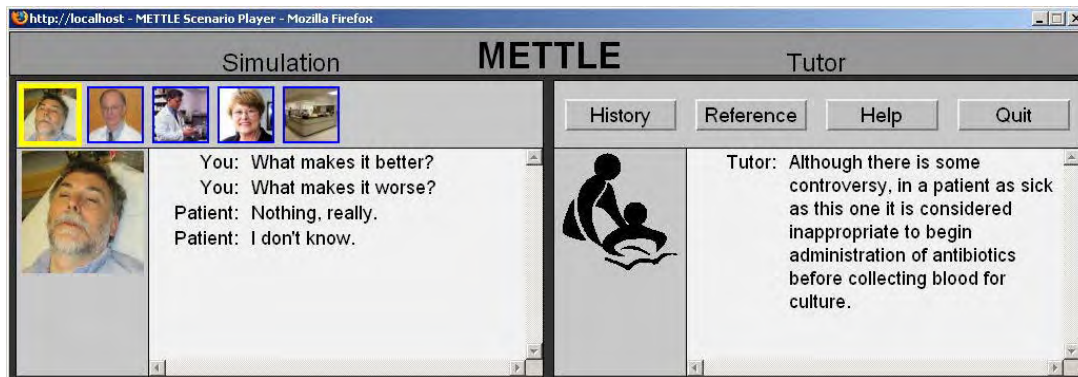


Figure 25. Tutor Reaction to Ill-Advised Student Action.

Assuming the student takes the hint implicit in the Tutor's comment, they might start looking for tests worth ordering. Figure 26 shows the menu that pops up in response to a click on the 'Test' button, offering several categories of tests. In response to selecting the 'Panels' item, the system pops up the Diagnostic Panels order-form window shown in Figure 27. Here we assume the student orders blood gases, a metabolic profile, and urinalysis. Note that while a full repertoire of tests is generally listed in these order forms, many of them may be disabled if the scenario authors did not think it worth providing test results for what they considered irrelevant tests. In similar manner to the negative feedback shown in Figure 25, Figure 28 shows the Tutor's positive feedback in response to the student's choice to order blood gases.

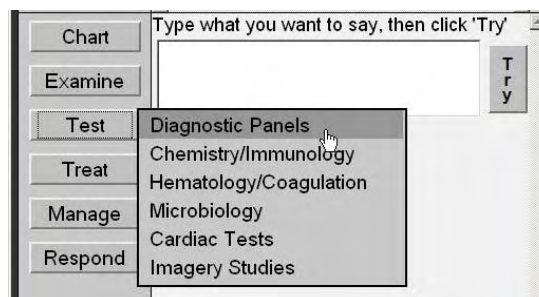


Figure 26. Pop-Up Menu of Treatment Categories.

Figure 27. Patient Test Order Form with "Diagnostic Panels" Options.



Figure 28. Tutor Reaction to Appropriate Student Action.

At this point, having taken a range of actions—and especially having ordered some tests—it is worth looking at the patient’s chart to see what kinds of information have accumulated there. Figure 29 shows the first of several chart pages: the Admissions cover sheet. By clicking on the tabs near the top of the window, the student can review other patient information. Figure 30 shows the Complaint page of the chart. Note that the final item on that page reflects the results of one of the interview questions asked by the student earlier in the interaction.

Patient Chart	
Account	123456789
Name	Ryan Smith
Street Address	123 Main Street
City	Boston, MA
Phone	617-555-1234
SSN	123-45-6789
DOB	9/18/1960
Age	48
Sex	Male
Marital Status	Married
Religion	Buddhism
Reason for Visit	Severe fever
Accompanied By	Wife
Acc. Date/Time	2/12/2008 18:47

Figure 29. Patient Chart – Admissions Tab.

Patient Chart		
Student	02/12/2007-18:50:00	His fever is now 102.2 orally and he is having sweats.
Student	02/12/2007-18:50:00	He now has chest discomfort on deep breath.
Student	02/12/2007-18:50:00	He is short of breath at rest.
Student	02/12/2007-18:50:00	Headache is worse.
Student	02/12/2007-18:50:00	Weakness is worse and the patient now feels unable to get out of bed.
Student	02/12/2007-18:50:00	His family feels that at times in the last few hours he seems slightly confused and uninterested in his surroundings.
Student	02/12/2007-19:08:20	Patient complains of coughs, night sweats, chills, headache, stomach and chest pain, and vomiting.

Figure 30. Patient Chart – Complaint Tab.

Figure 31 shows the chart's Exam page. Like the Complaint page, its contents is a mixture of findings entered as part of the patient's admission—that is, information reflected in the scenario briefing—plus information uncovered by the student's actions. In this case, the last two items “Eyes clear” and “Nose not running” are results of the interactive examination.

http://localhost - METTLE Patient Chart Window - Mozilla Firefox

Help

Patient Chart

Close

Admission

Complaint

History

Exam

Vitals

Tests

Treatments

Student	02/12/2007-18:50:00	Constitutional: Patient is slightly overweight. He looks ill and does not greet the examiner. The patient lies in bed looking at the ceiling or keeps his eyes closed.
Student	02/12/2007-18:50:00	Skin: Normal.
Student	02/12/2007-18:50:00	HEENT: The head is normocephalic and atraumatic. The neck seems slightly stiff and the patient has pain on movement of the neck.
Student	02/12/2007-18:50:00	Chest: The chest shows slight rales diffusely.
Student	02/12/2007-18:50:00	Heart: Regularly irregular. Heart rate slightly rapid. There is no murmur. Heart sounds are normal.
Student	02/12/2007-18:50:00	Abdomen: Normal exam.
Student	02/12/2007-18:50:00	Extremities: Extremities are slightly cool. There is very slight cyanosis of the nail beds.
Student	02/12/2007-18:50:00	Neurological: Patient is well oriented when pressed and there is no focal neurologic abnormality. However the patient at times seems uninterested and does not always answer questions. He has neck stiffness as noted above.
Student	02/12/2007-19:08:35	Eyes clear
Student	02/12/2007-19:08:36	Nose not running

Figure 31. Patient Chart – Exam Tab.

Figure 32 shows the chart's Vitals page, containing a table with one row for each set of vitals measurements that are taken. The vitals would generally be recorded at the start, and then additional sets of measurements can be scheduled to occur at later times during the scenario run (or optionally in response to explicit actions taken by the student).

Patient Chart						
Admission	Complaint	History	Exam	Vitals	Tests	Treatments
Time	Temp	Heart Rate	Resp. Rate	BP	O2 Sat.	Weight
02/12/2007-18:50:00	102.2 (tympanic)	118	28/min	115/75	93 (room air)	195 lbs

Figure 32. Patient Chart – Vitals Tab.

Figure 33 shows the chart's Tests page, showing the results of tests ordered by the student. This is perhaps the most important page of the chart, as it is the only page that displays information not available elsewhere in the system (e.g. in the initial briefing, or in the incremental results of interviews or examinations). In Figure 33 we see the results of the Urinalysis that we ordered in Figure 27; scrolling off the bottom of the page we see additional results from the Basic Metabolic Panel, and further down would be the results of the Arterial Blood Gases test. Test results are presented in formats similar to those used in normal practice—here a table with one attribute on each row, and the two main columns comparing data from this patient against reference values, with out-of-range values highlighted (here using a yellow background). An important point to note is that, while it is possible to script any amount

of simulated delay between the time a student orders a test and the time the result appears in the chart, for purposes of keeping this sample scenario moving quickly, we have chosen to have test results appear immediately. In some cases those results may explicitly note that they would normally only be available at 12 or 24 hours (e.g. cultures). In some cases the results are explicitly incomplete (e.g. provisional 12 hour results that might be updated if we could wait 24 hours).

Patient Chart			
Admission Complaint History Exam Vitals Tests Treatments			
Student	02/12/2007-19:09:51	Order Urinalysis Complete	
		Color	yellow yellow
		Clarity	Clear Clear
		GLUCOSE, URINE	NEGATIVE NEGATIVE
		BILIRUBIN, URINE	NEGATIVE NEGATIVE
		KETONE, URINE	NEGATIVE NEGATIVE
		SPEC GRAV URINE	1.020 1.002-1.035
		OCC BLD URINE	NEGATIVE NEGATIVE
		PH URINE	5.0 5.0-8.0
		PROTEIN, URINE	NEGATIVE NEGATIVE
		NITRITE, URINE	NEGATIVE NEGATIVE
		LEUKOCYTE ESTER	NEGATIVE NEGATIVE
		WBC/HPF	RARE 0-2 0-4 PER HPF
		RBC/HPF	RARE 0-2 0-2 PER HPF
		BACTERIA	<10 0 PER HPF
		CRYSTALS	PRESENT NONE SEEN
		CASTS	PRESENT NONE SEEN
		SQUAMOUS EPIS	0-2 0-2 PER LPF
		AMORPHOUS MATER	PRESENT NONE SEEN
		COARSE GRAN CAST	FEW 3-4 0 PER LPF
		FINE GRAN CAST	FEW 3-4 0 PER LPF
		HYALINE CAST	FEW 1-2 0-2 PER LPF
Student	02/12/2007-19:09:51	Order Basic Metabolic Panel	
		Na	142 135-144
		K	4.4 3.5-5.1

Figure 33. Patient Chart – Tests Tab.

To complete our tour of the patient chart, Figure 34 shows the Treatments page. Here we see a record of all the treatments ordered using the various order forms available under the ‘Treat’ button on the main Player screen.

Patient Chart			
Admission Complaint History Exam Vitals Tests Treatments			
Student	02/12/2007-19:08:48	Order Normal Saline IV Drip	
Student	02/12/2007-19:08:48	Order IV Placement	
Student	02/12/2007-19:08:56	Order IV amikacin	

Figure 34. Patient Chart – Treatments Tab.

At this point, the student might again ask the Tutor for a hint. Figure 35 shows the result of clicking the 'Hint' button at this point in scenario play: a suggestion that imagery might help reveal what is causing the patient's breathing difficulties. In Figure 36 we see the student ordering a Chest X-Ray from an order form available in response to selecting the 'Test' button's Imagery Study menu option. Here the student orders a Chest X-Ray.



Figure 35. Asking for Another Hint.

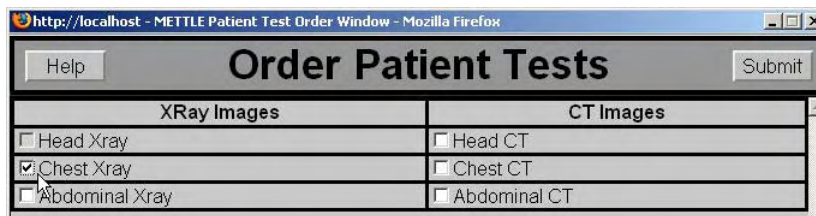


Figure 36. Ordering Imagery Studies – Chest X-Ray.

Figure 37 shows the Tutor's positive feedback response to the Chest X-Ray order. Most interestingly, Figure 38 shows the chart's Test page again, this time with the results of the Chest X-Ray displayed. In general, test results are HTML displays, and can easily embed imagery along with tabular data or textual descriptions. It would even be possible to embed sound or video files if they would do a better job of communicating important results.



Figure 37. Positive Feedback on Chest X-Ray Order.

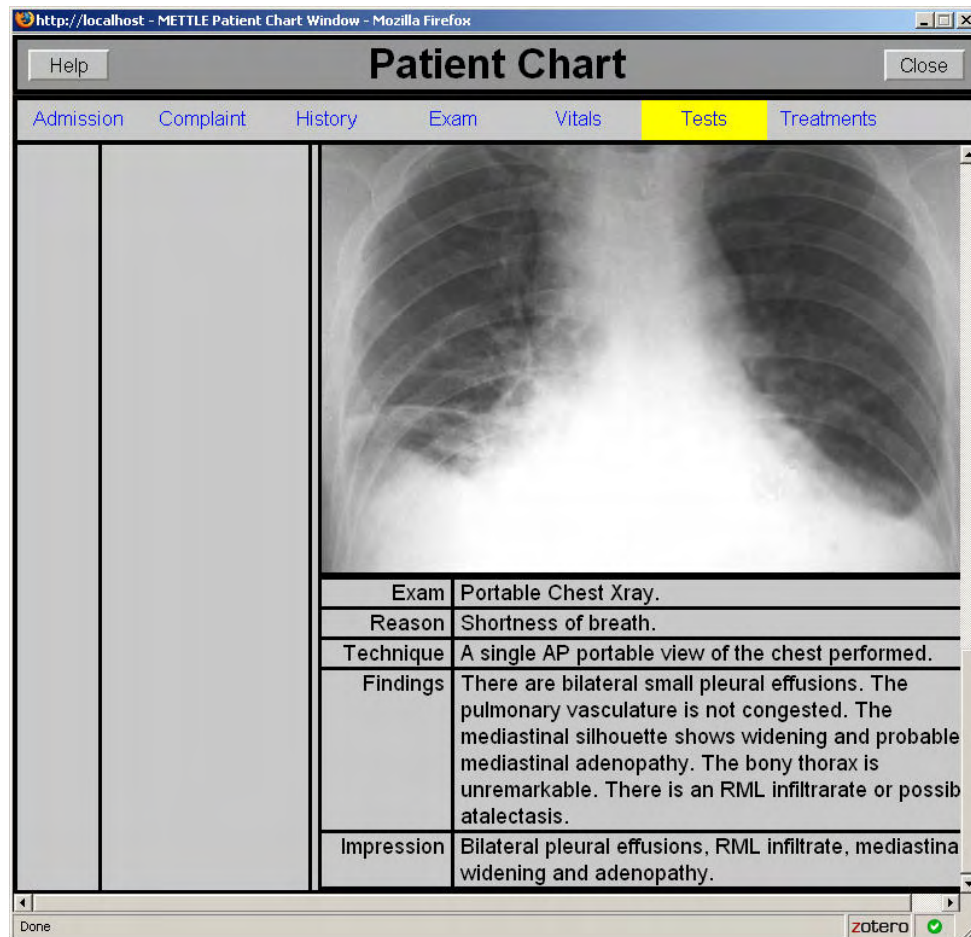


Figure 38. Patient Chart – Results of Chest X-Ray.

In general, this scenario is quite flexible with regard to the actions the student might take at any time. At this state of play there are several ways things might go. First, the 'Hint' and related buttons can be used guide a student through a reasonable sequence of further actions. Second, even if the student does not ask for hints, eventually the Tutor will offer some suggestions based on the passage of time and key actions left undone. Third, the student may take any of a range of actions that suggest they are either coming to some conclusions, or running out of steam in their diagnostic explorations, in which case the Tutor may launch into a more extended dialog to review the state of their diagnostic thinking.

The following figures show a sequence based on the second option above: the Tutor offers a proactive prompt effectively reminding the student of information that was in the initial briefing, and the student follows up the lead. Figure 39 shows the prompt in the Tutor area, and the student in the middle of asking the suggested question. Figure 40 shows the student in the middle of following this advice: they have asked the suggested question, been told (reminded) about the cousin who went to Memorial Hospital, and are in the middle of following up by asking the cousin's name.

In Figure 41 the student has clicked on the icon representing the Memorial Hospital ED in preparation of following up the lead about the sick cousin (note that a request for hints can prompt the student to take this action if they do not think to do it themselves, or do not know how to use the actor icon pallet). At this point, however, the Tutor has decided this shift in the student's focus is a good time to assess where they have gotten to in their diagnostic thinking. Thus over on the right side of Figure 41 the Tutor has initiated one of its extended diagnostic rationale dialogs. It starts with a simple yes/no (multiple choice) question asking the student whether they feel they know enough to settle on a diagnosis.



Figure 39. Tutor Proactive Prompt Followed-Up by Student.

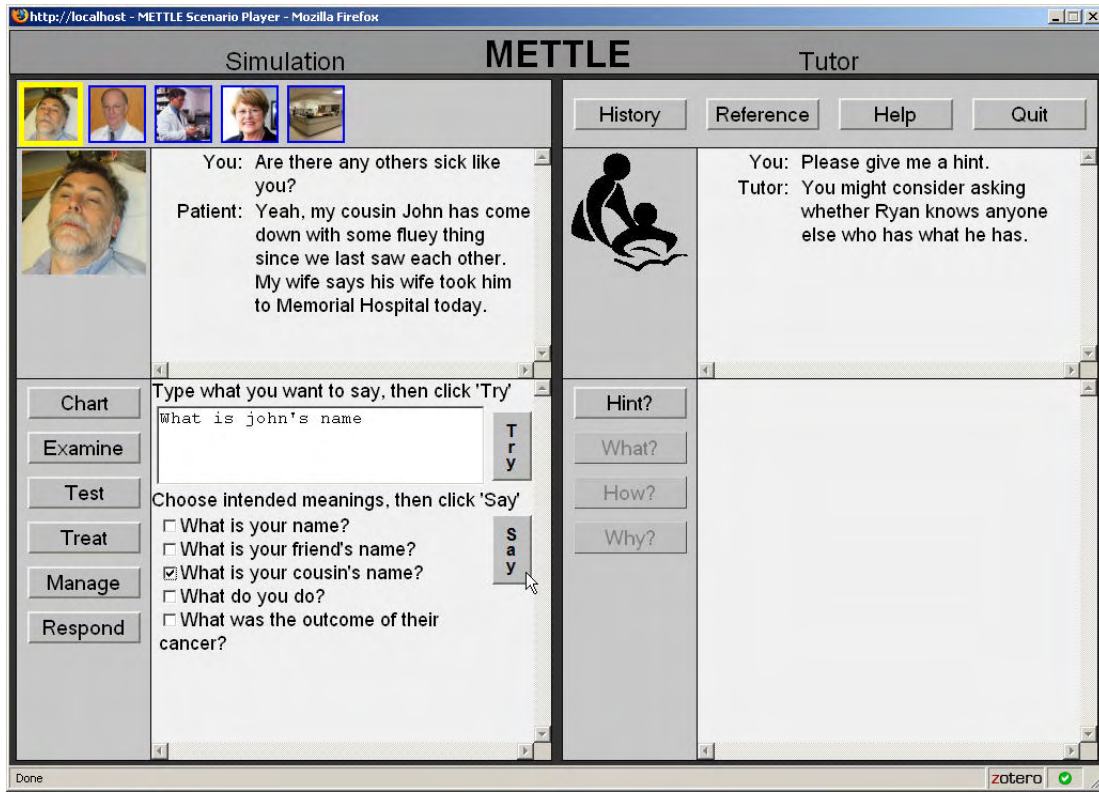


Figure 40. Student Reminded of Sick Cousin.

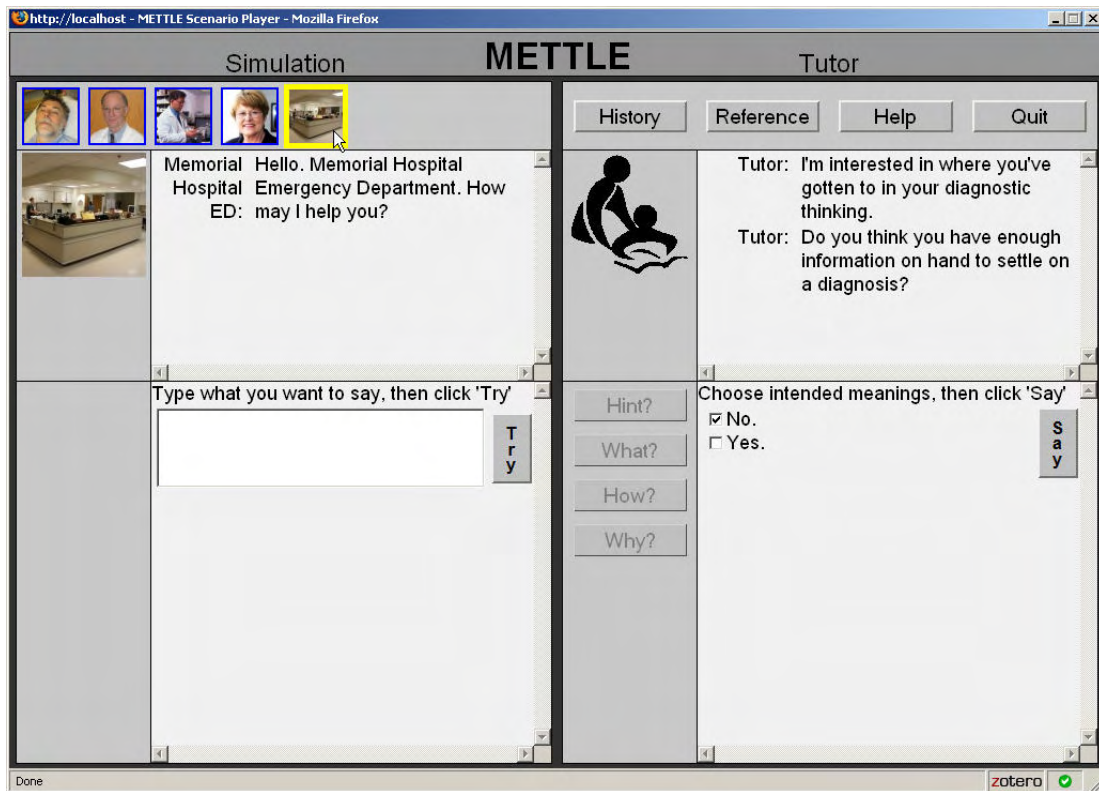


Figure 41. Student Calls Memorial ED and Tutor Launches Diagnostic Rationale Dialog.

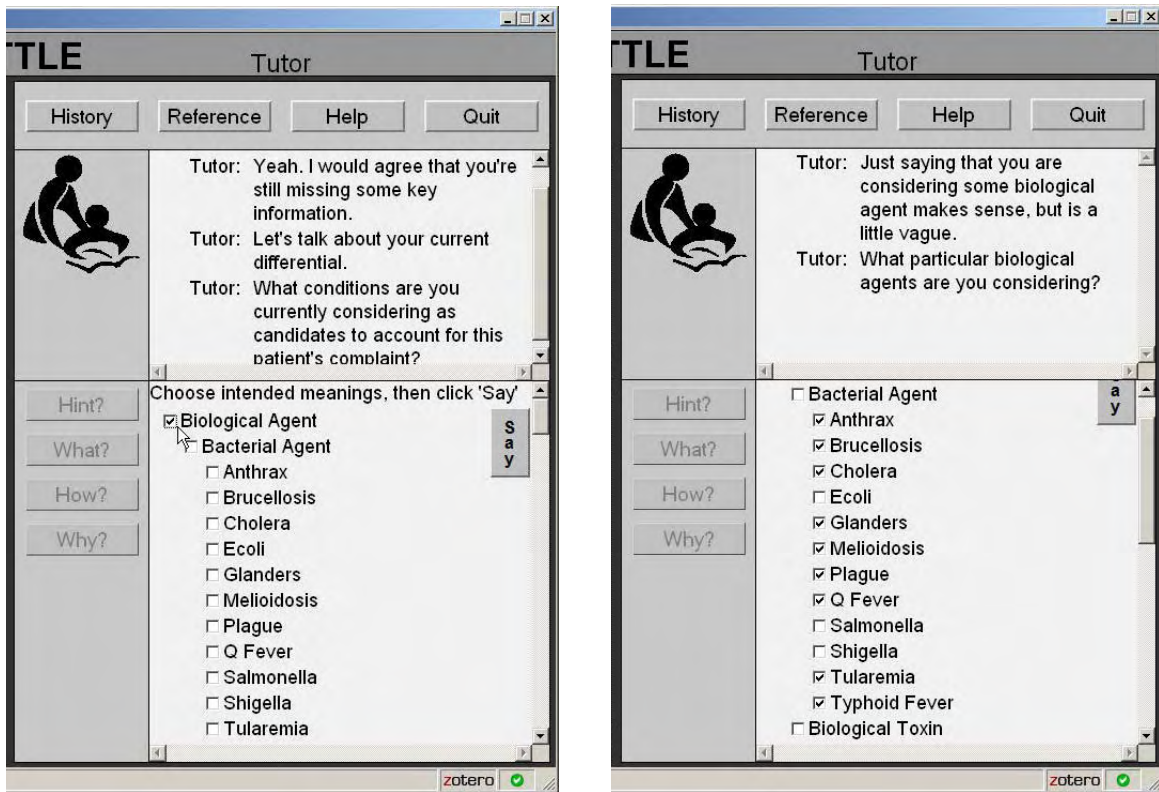


Figure 42. Tutor Elicits and Refines Student's Current Differential Set.

Though the chest X-Ray is important evidence, it is still too early for the student to have arrived at a firm diagnosis. The left side of Figure 42 shows the Tutor's agreement to the student's answer of "No" to the dialog's kick-off question, followed by the transition into prompting for elements of their current differential. The range of possible student responses is pre-enumerated as a large tree of options: there are about 100 CBR agents, plus about 2-dozen conventional conditions, and there are options at different levels of specificity as well (e.g. "Biological Agent," versus "Bacterial Agent," versus specific bacterial Agents like "Anthrax," "Brucellosis," etc. We have found it useful to adopt this middle ground between totally free-response and short multiple-choice questions, because; it helps keep the student within the envelope of useful system interaction without providing too much directive prompting.

Appendix F (in particular Section 12.2), despite showing a variant of this dialog (one designed for a point where the student has collected more evidence), gives a sense of how the dialog is put together, and the way the Tutor can repeatedly prompt for more, or more specific, elements in the differential. For instance, the right half of Figure 42 shows the Tutor returning with a critique of the student's suggestion that they are considering "Biological Agents," and a prompt for more specific information.

As an aside, we note that METTLE's Reference section includes a tab for Conditions (including CBR Agents) that matches the taxonomy of possible responses to the Tutor's diagnostic questions. At any time, the student is free to explore the available information. For instance, Figure 43 shows the results of clicking the Scenario Player's 'Reference' button and then selecting Brucellosis. The system has links to three different sources of information on Brucellosis: a CDC web site, and sections from two different military PDF documents (Army FM 8-284, and the Textbooks of Military Medicine volume on *Medical Aspects of Chemical and Biological Warfare*). Figure 44 shows a Textbook chapter on Brucellosis available for rapid review

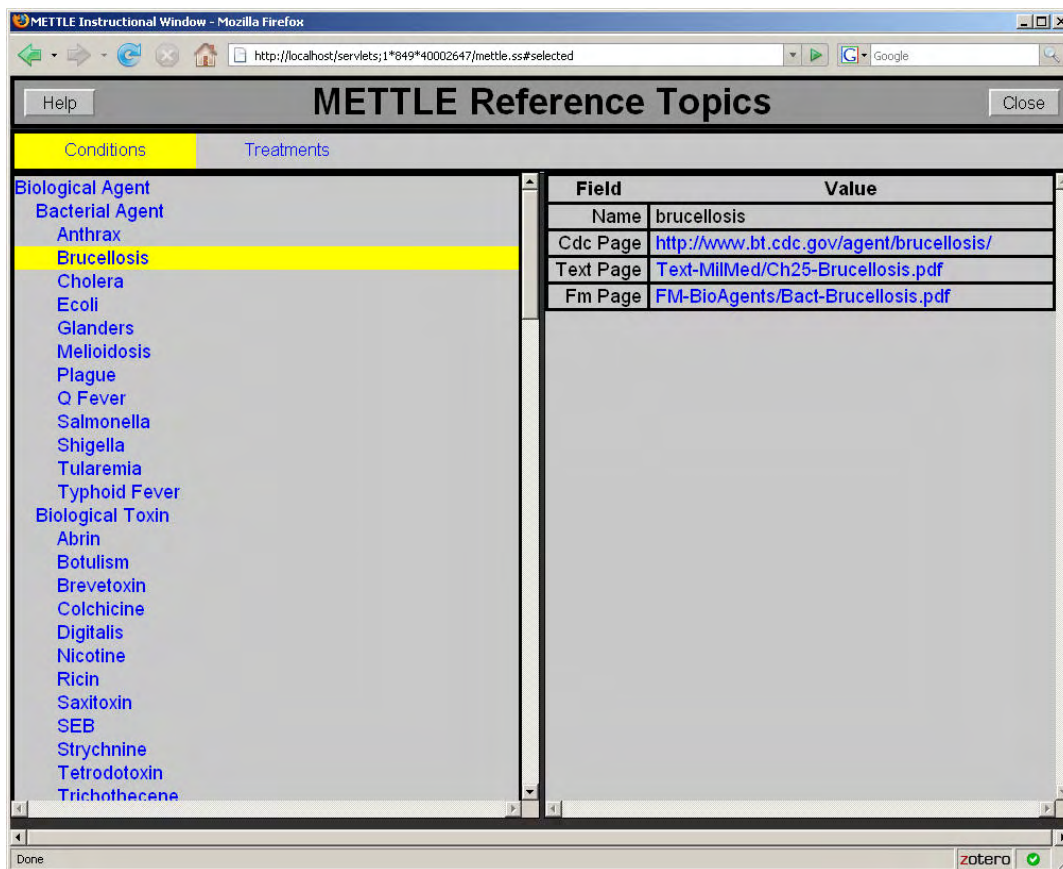


Figure 43. METTLE Reference Section, Conditions Tab with Focus on Brucellosis.

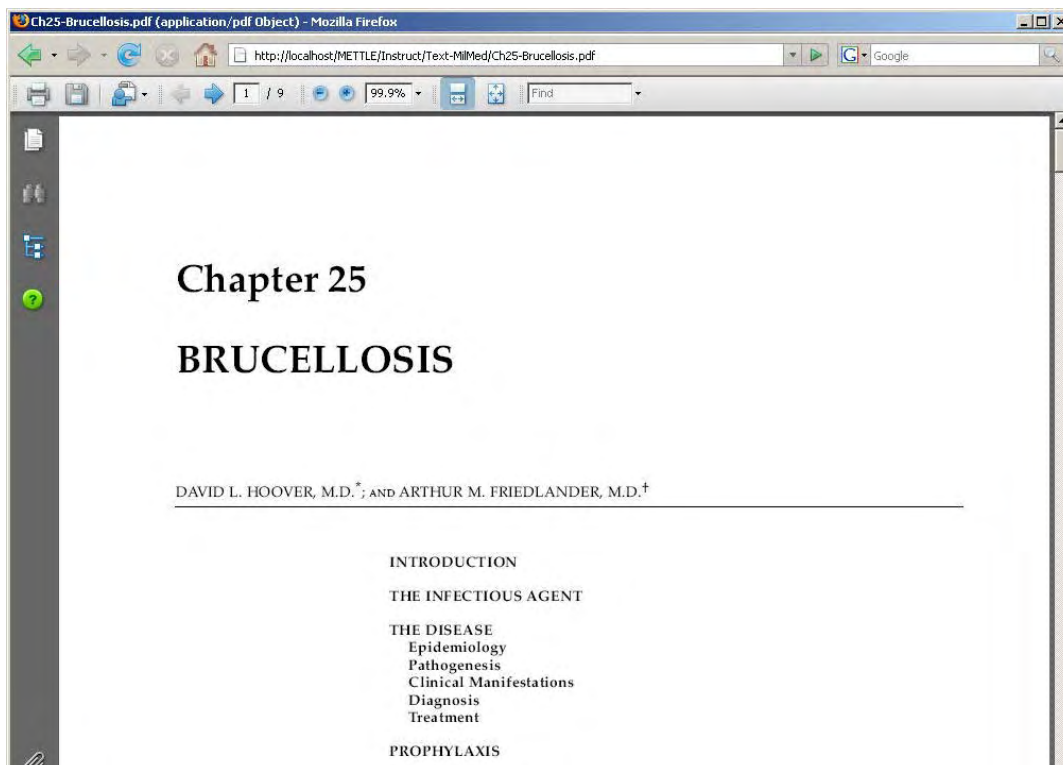


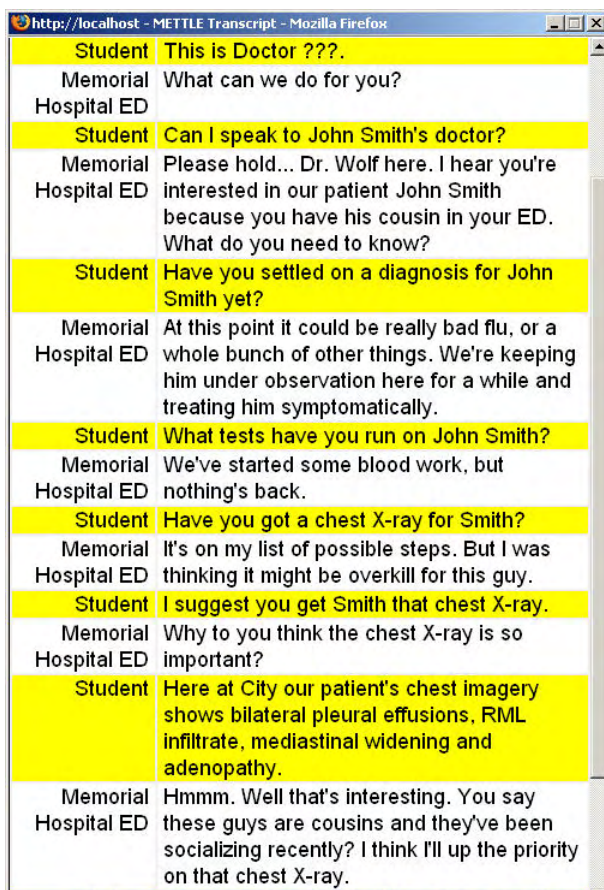
Figure 44. Link from METTLE References List to Authoritative Information Sources.

The right half of Figure 42 showed the student responding to prompting for more specificity with a mostly reasonable roster of possible conditions. Figure 45 shows excerpts from the transcript window's summary of the Tutor's response to these selections. While many of these conditions are worth maintaining as possibilities, the Tutor notes expected findings for Cholera and suggests the student check the consistency one of those against the condition of the current patient.

Tutor	Sure, based on chest imagery, anthrax could be looking like a stronger possibility.
Tutor	Brucellosis has been known to produce pulmonary effects that look a lot like TB, so it's not unreasonable to keep brucellosis in the running at low probability.
Tutor	Cholera is characterized clinically by profuse watery diarrhea, vomiting, circulatory collapse and shock. Check for reports on this patient's stools.
Tutor	Chest imagery provides some additional evidence that is consistent with glanders. In pulmonary infections, pneumonia, pulmonary abscesses, and pleural effusion can occur. Chest X-rays will show localized infection in the lobes of the lungs.
Tutor	Chest imagery is reasonably consistent with a pulmonary melioidosis infection. This form of the disease can produce a clinical picture of mild bronchitis to severe pneumonia, which would account for many of the imagery findings, though not the widened mediastinum. The onset of pulmonary melioidosis is typically accompanied by a

Figure 45. Some Tutor Critiques of Elements in Student's Differential.

Figure 46 picks up the thread of the scenario interaction, this time summarizing a series of text-based interactions with the simulated Memorial Hospital ED as a transcript fragment. Here we see the student appropriately both seeking information that might inform his diagnosis of Ryan Smith, as well as sharing information that might help Memorial in its handling of John Smith.



Student	This is Doctor ???.
Memorial Hospital ED	What can we do for you?
Student	Can I speak to John Smith's doctor?
Memorial Hospital ED	Please hold... Dr. Wolf here. I hear you're interested in our patient John Smith because you have his cousin in your ED. What do you need to know?
Student	Have you settled on a diagnosis for John Smith yet?
Memorial Hospital ED	At this point it could be really bad flu, or a whole bunch of other things. We're keeping him under observation here for a while and treating him symptomatically.
Student	What tests have you run on John Smith?
Memorial Hospital ED	We've started some blood work, but nothing's back.
Student	Have you got a chest X-ray for Smith?
Memorial Hospital ED	It's on my list of possible steps. But I was thinking it might be overkill for this guy.
Student	I suggest you get Smith that chest X-ray.
Memorial Hospital ED	Why to you think the chest X-ray is so important?
Student	Here at City our patient's chest imagery shows bilateral pleural effusions, RML infiltrate, mediastinal widening and adenopathy.
Memorial Hospital ED	Hmmm. Well that's interesting. You say these guys are cousins and they've been socializing recently? I think I'll up the priority on that chest X-ray.

Figure 46. Transcript of Conversation with Memorial Hospital.

The scenario would continue in this vein, with the student pursuing a mix of multi-modal information gathering, and interaction with simulated characters, possibly guided by additional Tutor hinting. This would culminate eventually in a version of the diagnostic rationale discussion that applies when the student has gathered sufficient evidence that they ought to be able to reach a correct diagnosis of anthrax.

Figure 47 shows a relatively early stage of that dialog, assuming the student has correctly identified anthrax as the most likely diagnosis. Here the Tutor is not simply accepting and critiquing the diagnosis; rather it is exploring the key evidence the student should have picked up on along the way, starting with evidence that was included in the patient's original presentation and complaint.

Figure 48 continues in like manner, but focusing on the microbiological results that are most important to the ultimate diagnosis. It also illustrates the Tutor following up a partial answer (i.e. the student cited the bacteria's gram stain, but not its shape). The first question/prompt pair is relatively open-ended, and the Tutor offers a fairly large space of potential findings as possible answers. The second question is far more focused, and accordingly the space of possible answers is narrowed to match.

Finally Figure 49 reviews the imagery findings, and the student correctly identifies mediastinal widening as the most distinctive and important imagery finding supporting a diagnosis of anthrax.

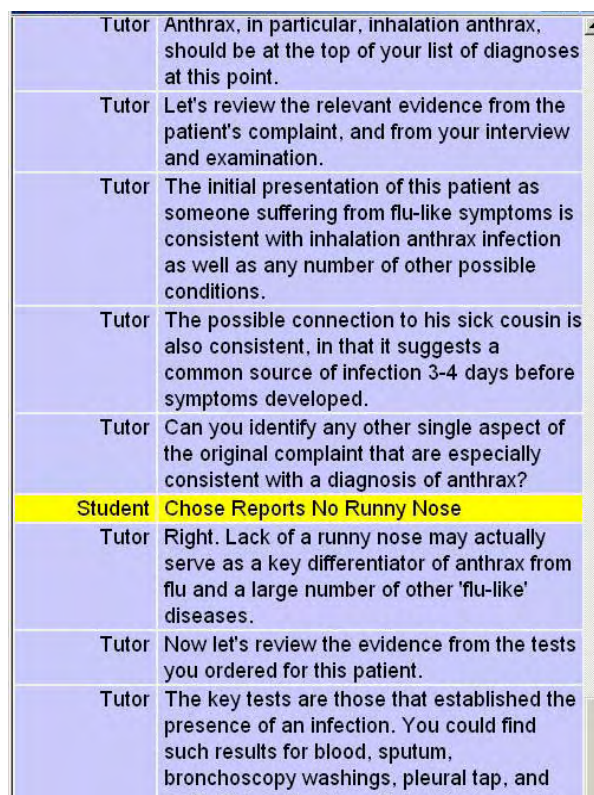
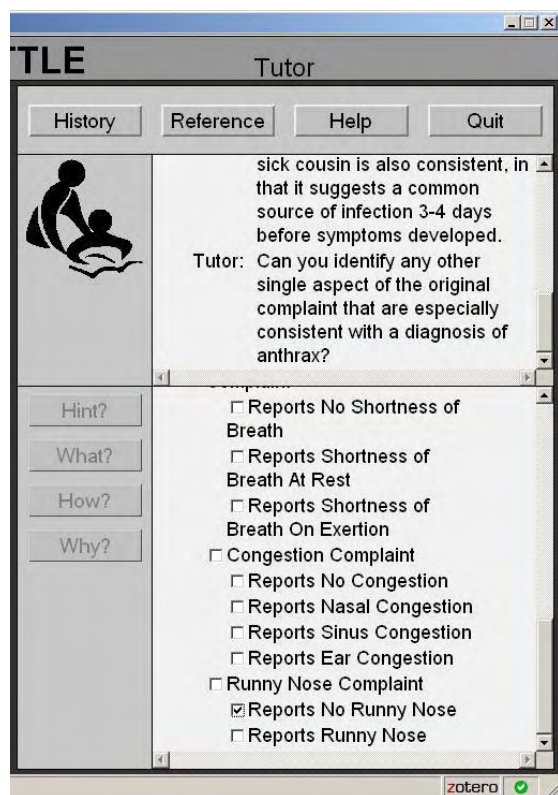


Figure 47. Final Diagnostic Rationale Discussion – Review of Key Patient Presentation Findings.

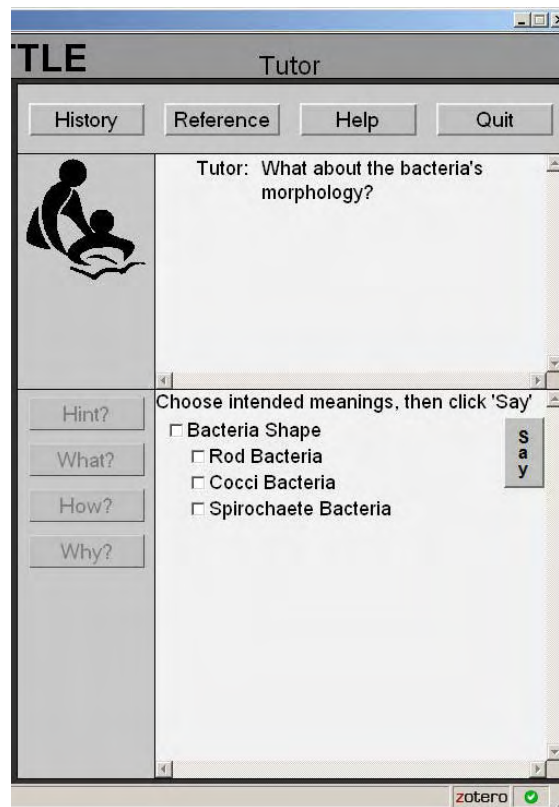
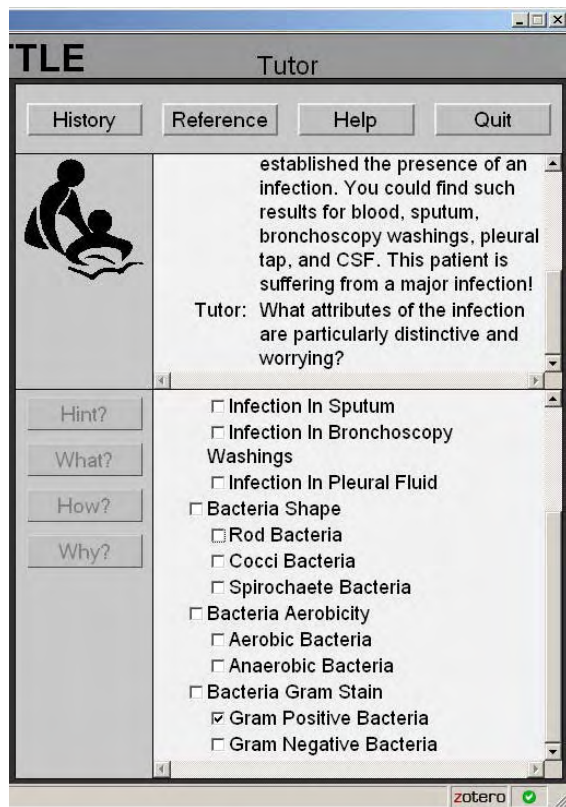
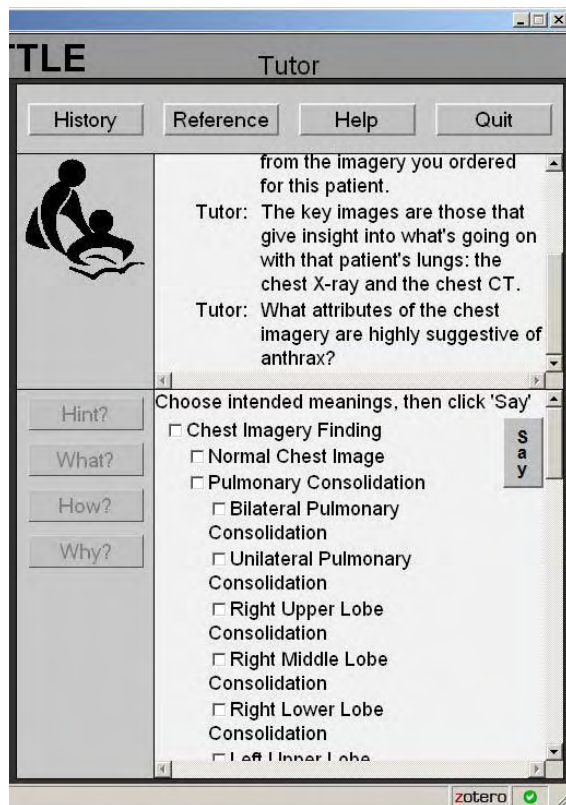


Figure 48. Final Diagnostic Rationale Discussion – Review of Key Microbiological Findings.



Tutor	Finally, let's review the evidence from the imagery you ordered for this patient.
Tutor	The key images are those that give insight into what's going on with that patient's lungs: the chest X-ray and the chest CT.
Tutor	What attributes of the chest imagery are highly suggestive of anthrax?
Student	Chose Mediastinal Widening
Tutor	You should remember that radiographic findings of pneumonia, mediastinal widening, and pleural effusions, in combination with other evidence, is very highly suggestive of inhalation anthrax.
Tutor	With the evidence in hand we should certainly be considering anthrax as a possible or even likely diagnosis. Given the seriousness of the disease for this patient, and potentially for the community we should probably: <ul style="list-style-type: none"> 1. Start aggressive treatment 2. Seek an informed second opinion and further confirmation 3. Start getting the word out to engage a larger scale emergency response.
Tutor	Based on this review, you should feel reasonably confident that you have a handle on what is most likely going on with this patient.

Figure 49. Final Diagnostic Rationale Discussion – Review of Key Imagery Findings.

The scenario ends with a final review of the key lessons by the Tutor. The topics covered include: (1) difficulties in arriving at a diagnosis of anthrax, (2) key indicators for establishing such a diagnosis, (3) treatment of patients with anthrax, (4) possible impacts on the ED, the hospital, and the wider community.

10 Appendix D: Authoring Tools

10.1 Content Types and Organization (Directory Structure)

The different fragments of content are organized in a directory structure that provides specific locations for specific pieces of content. In this section we document that directory structure, largely as a way of providing a complete map of the kinds of content that need to be developed. Since METTLE is a web application, the directory root for all its content is known as `web-root`. Following conventions of the web server, the content beneath that root is split into two main parts: `web-root/htdocs` contains all the standard HTML (and related CSS, JavaScript, and media) files that are served up as standard browser fare; `web-root/servlets` contains all the more specialized files defining content processed by the Concept, GRAIN, Discuss, and Enact layers. A third directory, `web-root/conf` contains server configuration information—a standard part of the web server distribution unaffected by METTLE.

Figure 50 extends this initial directory tree another couple of levels, showing the METTLE directories under `htdocs` and `servlets`. Like the `conf` directory, the `lib` directory is a relatively standard part of a web server, in this case holding reusable images and JavaScript files; we will not discuss it further.

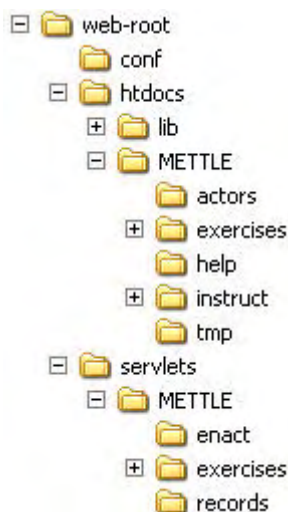


Figure 50. Top Levels of METTLE Content Directory Structure.

The directory `web-root/htdocs/METTLE` contains five sub-directories. (1) The `actors` directory is the home for reusable image files associated with simulated actors; each such actor must have three image files: (a) a thumbnail (50x50 pixel) image, (b) a gray-out version of the thumbnail, and (c) a slightly larger image (116x116 pixels) for when they are the currently selected actor. (2) The `exercises` directory is the home for all exercise-specific multimedia content, and will be discussed further below. As might be expected, (3) the `help` directory contains the HTML pages that provide help in using the system, (4) the `instruct` directory contains HTML (and other media) pages that constitute the system's instructional materials, and (5) the `tmp` directory contains temporary files that get generated during runs of the system (the only such files at this point are sound files that concatenate multiple simulated character answers into one longer utterance as needed).

The directory `web-root/servlets/METTLE` contains three sub-directories. (1) The `enact` directory contains servlet data files that are used across all exercises: (a) default scripts, and (b) form specifications. (2) The `exercises` directory contains servlet data files that are specific to particular exercises and will be discussed further below. (3) The `records` directory accumulates individual student records; these are not authored, and so will not be discussed further. This METTLE directory also

contains two data files directly: the `courses` file that defines GRAIN layer structures, and the `concepts` file that defines Concept layer structures.

Beyond the `courses` and `concepts` files, then, the main authored content is distributed in files beneath the two `exercises` directories (under `htdocs` and `servlets`), which contain sub-directories for each exercise (METTLE scenario).

Figure 51 shows the `htdocs` directories for a scenario called “CBR-1-Anthrax.” Remember `htdocs` contains static web media, so scenario directories under `htdocs` contain scenario-specific media. Most of that media is organized into a set of sub-directories whose names correspond to the names of actors in the scenario; here we see a single sub-directory named `patient` that contains the scenario-specific media associated with the patient character. While any actor can have a `speak` directory full of sound (.wav) files, actors who can be medically *examined* and *tested* (i.e. patients) also have directories full of examination images (.jpg), and test results (.html) files (possibly with supporting media such as .jpg files, as for X-ray images). Also note that actor media files can be organized into a more elaborate directory tree reflecting available scenarios scenes and actor states if the actor has any media that varies by scene or state (e.g. if their test results would be different in a scene covering the first day of their illness than in another scene covering the second day of their illness). In addition to all of this actor-specific media, there can be a set of .html files at the top level containing the scenario and scene-specific briefings.

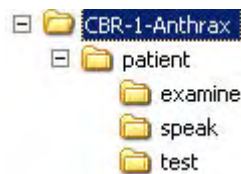


Figure 51. Exercise-Specific Directory under `htdocs`.

Figure 52 shows the contents of the `servlets/METTLE/exercises/CBR-1-Anthrax` directory—the program-specific data files for the particular scenario under discussion here. There are basically six types of files in this directory:

1. **The exercise file:** The exercise data file for an Enact scenario describes the overall shape of the scenario. This includes (a) the scenario name, (b) the scenario start time, (c) the set of scenario scenes (each with its own name and start time), and (d) the scenario’s cast of characters. Each actor in the cast must be assigned three image files (thumbnail, grayed-out-thumbnail, and larger image). Optionally, actors may also be assigned a set of possible states and a starting state, a set of configurable interaction modes (e.g. for patients, the chart, examination, testing, treatment, etc. pages), and sets of initial chart entries.
2. **dialog_files:** A set of files containing specifications of agent-initiative dialogs (Discuss layer scripts), whose names conventionally start with `dialog`, followed by the actor’s name, followed by a unique dialog name (all separated by underbar characters).
3. **script_files:** A set of files containing specifications of agent behaviors (Enact layer scripts), whose names conventionally start with `script`, followed by the actor’s name, followed by optional scene and state names (again all separated by underbar characters; here we add the underbars even when the script file is not specialized to a particular scene or state to emphasize the contextual structure, and to ensure a clear interpretation in the case of script files specialized to a particular state but applying to any scene).
4. **spread_files:** A set of files containing specifications of agent behaviors (Enact layer scripts) in exactly the same format as the `script` files. In fact the only difference between `spread` files

and `script` files is that the `spread` files are automatically generated from a spreadsheet. We adopt a slightly different naming convention so that both sets of files can be maintained without over-writing one another, because, as noted, the spreadsheet format is designed to make it easy to encode a common but limited subset of all possible behavior rules; thus a single actor might reasonably have both spreadsheet derived behaviors and some more complex scripted behaviors.

5. **Spreadsheet files (.ods):** The set of spreadsheet files used to generate the `spread` files described above. Each spreadsheet file's name must match the name of the actor whose behaviors it is intended to generate. Each spreadsheet file may contain any number of sheets, whose names follow a similar naming convention to script files (e.g. possibly including optional scene and state names, separated by mandatory underbar characters). Each sheet in these spreadsheets must adhere to a specific format (see 10.3 below) which is best insured by copying an existing file (or sheet) and replacing its contents.
6. **Spreadsheet generation files (.gen):** The spreadsheet files are used to automatically generate the `spread` files. The generation file is simply a convenience to automatically maintain a time-stamp that lets the system know when it last generated those `spread` files, so it can quickly check against the latest write-date of the corresponding spreadsheet file and determine if it needs to re-generate the `spread` files.

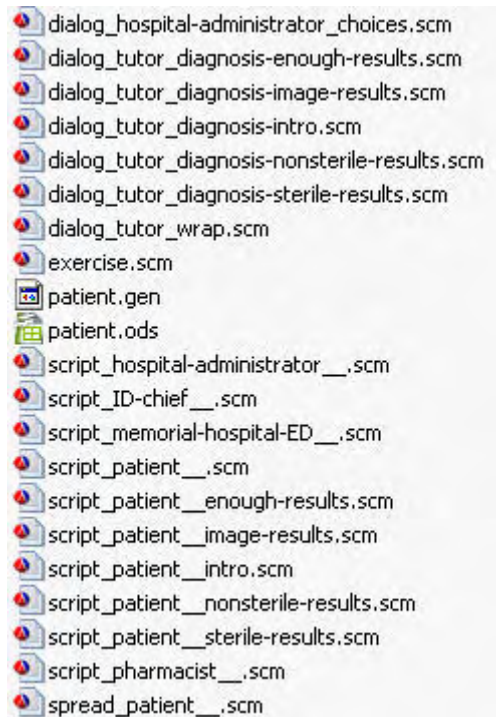


Figure 52. Exercise-Specific Directory under `servlets`.

In summary, there are six special format data files required by the METTLE system:

- **Courses:** Contains *GRAIN* level data defining a set of courses and associated curricula.
- **Concepts:** Contains *Concept* level data defining reusable taxonomies.
- **Form:** Contains *Concept* level data defining a specialized HTML forms for choosing concepts.
- **Exercise:** Contains *Enact* level data outlining the shape of an entire scenario.
- **Script:** Contains *Enact* level data for one agent (in one scene and state).
- **Dialog:** Contains *Discuss* level data for one extended dialog.

The current METTLE implementation offers some specialized tool-based authoring support for four of these six. It currently has no support for editing **forms**, in part because they are expected to change relatively rarely, and in part because the format is not terribly complicated. As noted earlier, it also has no specialized **dialog** editing tools, in this case primarily due to frustration with earlier generations of such tools that proved somewhat cumbersome; this is an area we will no doubt revisit in future projects. Meanwhile, as also noted, our ultimate decision to ensure that *all* of these data types have relatively straightforward textual file formats, combined with the continuing evolution of validation methods (e.g., as integrated into the Interpreter framework) guarantees that all aspects of the system remain authorable using COTS text editors.

10.2 Web-Based Authoring Tools

METTLE's web-based authoring tool suite primarily provides support for viewing and editing **courses**, **concepts**, and **exercise** files. Secondly, it also supports creating and editing the briefing html files. The first sequence of screen shots below focuses on the **courses** (or GRAIN) editing page. As with all of the web-based editing pages to follow, the GRAIN Editor was designed based on the earlier GRIST CustomEdit authoring tools. The general pattern, as shown in Figure 53, calls for a set of tabs across the top of the screen (here "Roles," "Courses," "Curriculum," "Exercises," and "Students"), supplemented by a few top-level buttons (here 'Concepts,' 'Help,' and 'Save'). The currently selected tab is highlighted in yellow, and other tabs can be selected by clicking on them. For each tab, the left half of the screen shows a collection of relevant objects—in Figure 53 it shows Roles—any one of which can be selected by clicking on it. The selected item is highlighted in yellow, and its details are displayed in the right half of the screen. A set of icon-buttons above the left column provides means to create, delete, or re-order the items in the current collection. A pair of icon-buttons above the right column provides means to accept or reject edits made to the currently selected item.

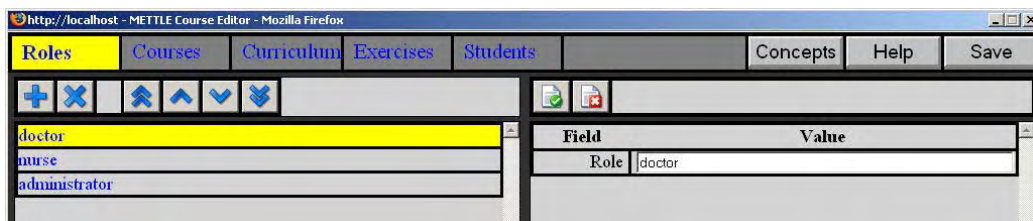


Figure 53. GRAIN Editor – Roles Tab.

Many fields, such as the names of elements, expect simple text strings. Some, such as the one that stores the set of Roles associated with a Course, expect lists of other elements. Figure 54 shows widgetry to support constrained list-picking, used in the case of Course Roles, and elsewhere.

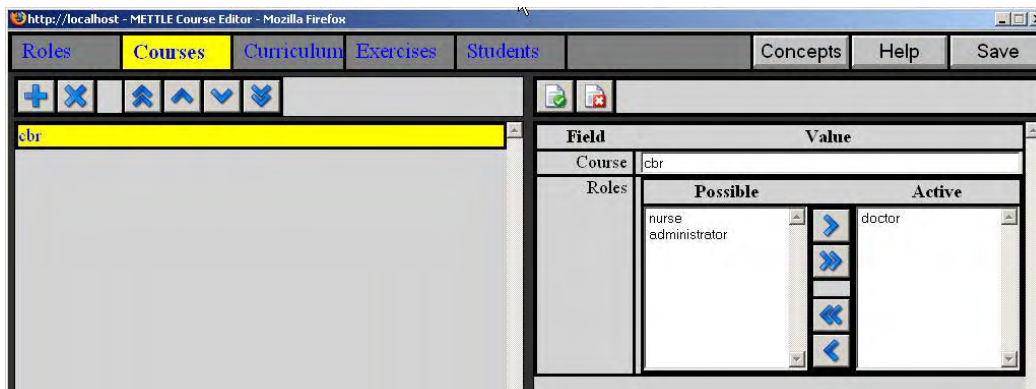


Figure 54. GRAIN Editor – Courses Tab.

Most of the collections are simple lists, however the curriculum space is organized into a tree structure. Figure 55 shows this slightly more complex kind of display, which requires some additional icon buttons to manage nesting of items relative to one another.

Field	Value
Point	ED-Practice
Description	General principles of good ED practice.
Parent	root

Figure 55. GRAIN Editor – Curriculum Tab.

The same kind of tree structuring applies to concept spaces as how in Figure 56 which captures the Concept Editor when focused on the Conditions concept space tab and the specific concept Anthrax. Obviously its form is isomorphic to the GRAIN Editor. As noted earlier, both are built on top of the Registry layer and its custom HTML widgetry.

Field	Value
Name	anthrax
Other Names	
Cdc Page	http://www.bt.cdc.gov/agent/anthrax/
Text Page	Text-MilMed/Ch22-Anthrax.pdf
Handbook Page	
Battlebook Page	
Fm Page	FM-BioAgents/Bact-Anthrax.pdf

Figure 56. Concept Editor – Conditions Tab.

Returning to the the GRAIN Editor, the Exercises tab shown in Figure 57 is notable mainly for the extra button in the details column that opens the Enact Exercise Editor page discussed further below.

Field	Value
Exercise	CBR-1-Anthrax
Title	Mystery Patient 1
Summary	In this scenario you will meet a patient who has arrived at your emergency department with an unknown illness. Your job is to diagnose and treat this patient, using the resources and teammates at your disposal.
Guidance	As you start the scenario, there will be a more detailed briefing. Once you get to the main simulation

Figure 57. GRAIN Editor – Exercises Tab.

Figure 58 completes our tour of the GRAIN Editor tabs. Normally new Students would be created using the “New User” page of the METTLE runtime environment, but the GRAIN editor provides an alternate central administration point for such data.

Field	Value						
Student	test						
Password	test						
Phone							
Email							
Roles	<table border="1"> <thead> <tr> <th>Possible</th> <th>Active</th> </tr> </thead> <tbody> <tr> <td>nurse</td> <td>doctor</td> </tr> <tr> <td>administrator</td> <td></td> </tr> </tbody> </table>	Possible	Active	nurse	doctor	administrator	
Possible	Active						
nurse	doctor						
administrator							

Figure 58. GRAIN Editor – Students Tab.

With Figure 59 we move to the Exercise Editor. The Scenes tab shown here enables creation, deletion, renaming, reordering, and assignment of start times to scenes of the current scenario.

Field	Value
Scene	scene
Start Time	02/12/2007-19:08:00
Briefing	Link to this scene's briefing

Figure 59. Exercise Editor – Scenes Tab.

Figure 60 shows the Actors tab, which provides mechanisms for editing the scenarios cast of characters. All of the attributes sketched earlier are editable from this screen.

Field	Value														
Actor	patient														
Image File	/METTLE/Actors/patient-116.jpg														
Live Icon	/METTLE/Actors/patient-50.jpg														
Gray Icon	/METTLE/Actors/patient-50-gray.jpg														
Setup	Link to this actor's setup														
All States	(intro nonsterile-results image-results sterile-results enough-results)														
Start States	<table border="1"> <thead> <tr> <th>Possible</th> <th>Active</th> </tr> </thead> <tbody> <tr> <td>nonsterile-results</td> <td>intro</td> </tr> <tr> <td>image-results</td> <td></td> </tr> <tr> <td>sterile-results</td> <td></td> </tr> <tr> <td>enough-results</td> <td></td> </tr> </tbody> </table>	Possible	Active	nonsterile-results	intro	image-results		sterile-results		enough-results					
Possible	Active														
nonsterile-results	intro														
image-results															
sterile-results															
enough-results															
All Modes	<table border="1"> <thead> <tr> <th>Possible</th> <th>Active</th> </tr> </thead> <tbody> <tr> <td></td> <td>chart</td> </tr> <tr> <td></td> <td>examine</td> </tr> <tr> <td></td> <td>test</td> </tr> <tr> <td></td> <td>treat</td> </tr> <tr> <td></td> <td>manage</td> </tr> <tr> <td></td> <td>respond</td> </tr> </tbody> </table>	Possible	Active		chart		examine		test		treat		manage		respond
Possible	Active														
	chart														
	examine														
	test														
	treat														
	manage														
	respond														

Figure 60. Exercise Editor – Actors Tab.

Figure 61 shows a pure JavaScript rich-text (HTML) editor embedded in the Exercise Editor web page, which provides a web-hosted way to create and edit briefings files for scenarios, and any scene that requires its own introduction.

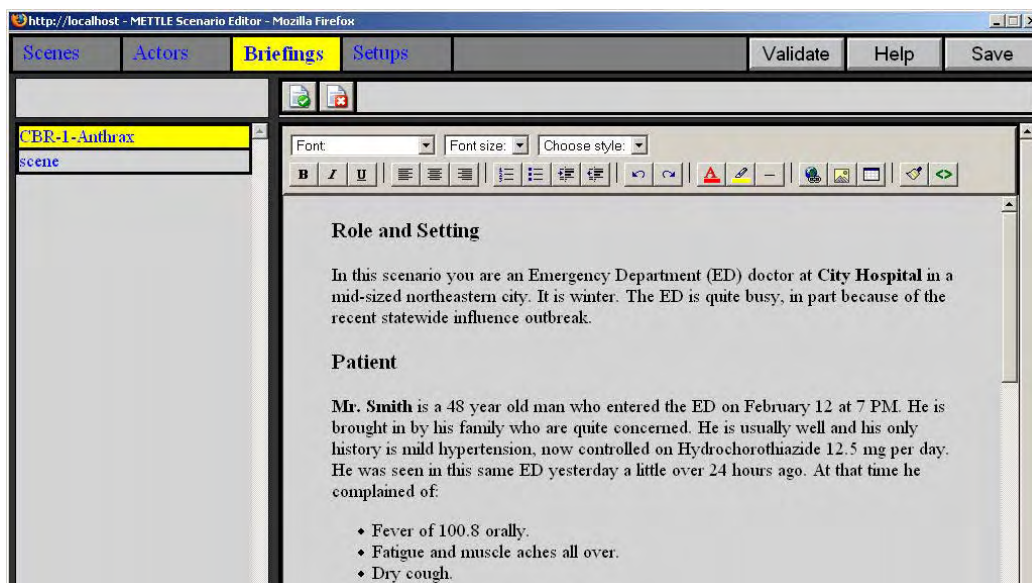


Figure 61. Exercise Editor – Briefings Tab.

Figure 62 shows the Setups tab, which provides a way to specify the setup actions to be associated with an actor. Currently, these setup actions only apply to patients; each of the 5 sub-tabs represents a part of the patient chart that can be seeded with data at the scenario's start. In general, it might prove useful to have arbitrary Interpreter actions executed for each actor at scenario or scene start, but this tool does not support that level of flexibility. Such capabilities depend on more detailed behavioral scripting of the kind enabled by the spreadsheets and text editing described in the following sections.



Figure 62. Exercise Editor – Setups Tab.

One additional important feature of the Exercise Editor is the 'Validate' button, which scans over all script files associated with the exercise and validates all Interpreter expressions found in all lines of all scripts. This is a first step towards the more pervasive and automated validation policy advocated earlier.

10.3 Spreadsheet-Based Behavior Authoring

The idea of using a commodity spreadsheet program to support high volume behavior authoring was prompted by the project's consulting domain researcher. Dr. Gilbert was familiar with spreadsheets and comfortable using them for simple tabular data collection and organization. Without prompting, she started building spreadsheets early on as she collected data on CBR conditions and diagnostic activities. When it came time to start specifying simulated actor behaviors, she expressed a desire to keep track of those in spreadsheet form as well.

On reflection, it becomes clear that spreadsheets have several strikingly important advantages over the more custom authoring tools we were developing: (1) everyone already knows how to use them with some level of comfort, (2) one of two or three different (but largely compatible) spreadsheet applications are already installed on most computers, and an excellent open source version (OpenOffice) can be freely installed on any machine that somehow lacks one, (3) they are designed to make it easy to scan relatively large amounts of data in a single display, and furthermore make it easy to print out nicely formatted versions of that data for detailed review away from a computer, (4) they support the common desktop computer usage idioms that users expect, such as block selection, cut-and-paste, undo, etc., (5) they run very quickly, hardly ever crash, and exist today, rather than next week.

Of course, spreadsheets also have several notable defects when it comes to trying to use them as ITS content authoring tools: (1) they don't inherently know anything about the specific formats of data the ITS expects or needs, so they can do only limited prompting and validation of data entry, (2) their forte is the two-dimensional table, so they lack strong natural mechanisms for handling nested structures and hierarchies, or even embedding open-ended lists, all of which are common in ITS content development, and (3) the leading proprietary spreadsheets generate files in arcane, complex, and often undocumented data formats.

Fortunately, those are defects we can work with in various ways. The *file format* issue is easiest to deal with. One possibility is simply to ask authors to remember to export their spreadsheet data to one of the common simplified exchange formats supported by all application—for instance CSV (or comma separated values). Our preference, however, was to avoid the need for special export where possible. Our choice of solution was prompted by the fact that some of the machines our consultants were using did not have MS-Office installed on them; instead the available spreadsheet was part of the free OpenOffice suite. OpenOffice uses an open standard file format, one based on XML. This means that it is relatively easy to develop parsers (and for that matter generators) for OpenOffice spreadsheet files. This is the approach we took. Authors on machines that have MS-Office can use Excel if they want to, though that now requires an import or export step to move between OpenOffice Calc and MS-Office Excel formats. Alternately, if they are going to be doing a lot of authoring, they can install the free OpenOffice suite.

With regard to the *prompting* defect, the first question is whether a tool that can do better prompting speeds up authors more than its other limitations (as compared to spreadsheets) slows them down. The second question is whether there might be ways to extend the range of prompts the spreadsheet can offer: spreadsheets make it easy to prompt for cells that must be filled with elements from a small fixed set (e.g. the universe of available Interpreter operators); with a bit more work, they can also prompt for small variable sets (e.g. the set of scenes or actors defined for a scenario); with large amounts of embedded scripting voodoo it might be possible to get them to offer more context-dependent prompts (e.g. the states appropriate to a particular actor in a scenario).

We have not gone very far down this road, because an approach to the *validation* defect seems likely to generate more rapid and portable payoff. While certainly helpful, prompting may not be critical when working over the kind of small sets that can be conveniently packaged into dropdown menus; the small sets do not present that large of a memory load. Prompting is potentially more useful when offering suggestions from very large sets; but in that case, a really good prompting user-interface—one that does not drown the user in useless suggestions—is tricky to design. In both cases, it may suffice to let the

author enter their best guess or memory of what they mean, and leave it to an external post-process to validate the input, notice mistakes, and suggest possible corrections. Such an external validator has the advantages in that it can be written in the main development environment (rather than some application-specific extension language) and it can access to all the rest of the ITS data (which is unlikely to be available to the spreadsheet).

All ITS data is unlikely to be available in the spreadsheet because the way to work around the data *representation* defects of spreadsheets is to adopt the dictum that tools should be used to the extent that they are helpful, and not to try to use spreadsheets for everything. The goal is to find the most common forms of minimal or fixed complexity structure that can be mapped into a spreadsheet without requiring undue effort on the part of authors (to understand and use the spreadsheet) or developers (to build the spreadsheet and accompanying translators).

All that being said, we have evolved a spreadsheet design that we believe can handle the vast majority of script line specifications likely to be needed in a scenario. We have done that by incorporating a relatively high but still fixed level of rule structure complexity, while ensuring that most of the time most of the fields can simply be left blank. As shown in Figure 63 the current METTLE spreadsheet format uses 26 columns: A-Z. We have set up the spreadsheet so that these column headers and the entire first three columns (A-C) remain in fixed positions, while the rest of the sheet content scrolls. Column C, or “Line Name” is the key to tracking what script line is being described by the sheet’s contents.

The figure consists of two screenshots of the OpenOffice.org Calc spreadsheet interface, showing the column headers for the Enact Behavior Spreadsheet. The top screenshot shows columns A through O, and the bottom screenshot shows columns A through Z.

Top Screenshot (Columns A-O):

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Script Line							Cue Test						Response Action
2	Run #	Line Name	L1	L2	N?	As	Op	Main-Arq	C1	C2	As	Op.	Arq.	Main-Arq

Bottom Screenshot (Columns A-Z):

A	B	C	P	Q	R	S	T	U	V	W	X	Y	Z
1	Script Line												
2	Run #	Line Name	Point	#	When	L1	L2	N?	As	Op	Main-Arq	Type	Tutor Comments Comment

Figure 63. Enact Behavior Spreadsheet Column Headers.

Figure 64 shows a snippet of a `patient.ods` file containing (partial) patient behaviors. This is a file that could appear in the `web-root/servlets/METTLE/enact` directory, meaning it contains specifications of *default* script lines; as noted earlier, for such defaults, it generally only makes sense to encode the common cue conditions, as the responses are likely to vary across scenarios (and possibly across scenes and states). The important point is that for a very common class of behaviors, authors need fill in only 5 columns of the spreadsheet, and in fact only two require any significant thought: the behavior name (in Column C) and the expected student utterances (in Column I). Note that blocks of spreadsheet rows are grouped together and interpreted as a single script line based on the appearance of a value in Column A (the horizontal block border lines are simply for human readability). So long as Column A contains either “once” or “many” (meaning the behavior can be used either only once, or as many times as triggered) the Row will be interpreted as the start of a new script line. Any other content in Column A will cause the line to be interpreted as a comment (e.g. as in Row 3).

Run #	Line Name	L1	L2	N?	As	Op	Cue Test	Main-Arg	C1	C2	As	Op	Arg	Main-Arg
once	Patient-Name	or				hear	What is your name?							
						hear	What do they call you?							
once	Patient-Birth	or				hear	When were you born?							
						hear	How old are you?							
						hear	What is your age?							
						hear	What is your date of birth?							
once	Patient-Referral	or				hear	Who sent you?							
						hear	Who recommended you come see us?							
						hear	Who told you to come to the hospital?							
once	Chief Complaint	or				hear	What seems to be the problem?							
						hear	What is bothering you today?							
						hear	What's troubling you today?							
						hear	What seems to be wrong?							
						hear	What is your current complaint?							
						hear	What brings you here today?							

Figure 64. Behavior Spreadsheet with Simple Default Line Cues.

Figure 65 shows a similar snippet of another `patient.ods` file, this time from the scenario-specific `web-root/servlets/METTLE/exercises/CBR-1-Anthrax` directory. This figure shows two complete script line specifications (including both cue and response) for behaviors that are completely specific to the particular scenario. While it will probably always make sense to be able to ask questions about a patient's name, age, referral, and complaint, it requires specific scenario circumstances to be worth expecting questions about a patient's cousin's health and name. This figure shows that for simple behaviors, an author need only fill in another two columns to specify the response part of a script line.

Run #	Line Name	L1	L2	N?	As	Op	Cue Test	Main-Arg	C1	C2	As	Op	Arg	Main-Arg
once	Others-Cousin-State	or				hear	How is your cousin?					say	My wife told me that his wife called to say he was heading over to the Memorial Hospital, so I guess pretty bad.	
						hear	Do you know how your cousin is doing?							
						hear	Is your cousin as sick as you are?							
once	Others-Cousin-Name	or				hear	What is your cousin's name?					say	My cousin's name is John Smith.	
						hear	Can you tell me your cousin's name?							
						hear	Who is the cousin you went to the game with?							
once	Others-Suggest-Call													ED-Dis

Figure 65. Behavior Spreadsheet with Simple Scenario-Specific Lines.

Figure 66 is included to demonstrate that while the spreadsheet makes it possible to specify simple behaviors in simple ways, we have included enough flexibility that it is also possible to specify much more complex behaviors. The script line "Others-Suggest-Call" uses the tutor behavior columns (P-Z) to pack a variety of tutor behavior into this patient's script. The Tutor-Test column values shown here check for whether the student has (a) asked this patient if they know of anyone else who is sick like them, or (2) asked specifically how this patient's cousin is doing, and (3) not yet asked Memorial Hospital's ED how this patient's cousin is doing. When those conditions are met, the basic Tutor hinting/explanatory comments (hint, what, how, and why) become active, with a priority ranking of 9 assigned to them (taken from Column Q, and controlling how those comments sort out versus other possibly active

comments). Ten minutes after those conditions are met (based on the value in Column R), if the conditions are still true (that is, if the student has not in the mean-time asked Memorial ED about the cousin) then the proactive tutor comment is delivered. If the student requests one of these tutor comments, or has the proactive comment thrust upon them, they will be debited for inability to demonstrate a skill associated with the curriculum point “ED-Diagnosis-Infection.”

Run	Line Name	Point	#	When	L1	L2	N?	As	Op	Main-Arg	Type	Tutor Comments
10	once Others-Suggest-Call	ED-Diagnosis-Infection	9	00:10:00	and	or			line	Complaint-Others	hint	Given this patient's condition, it's interesting that other people he was with may have the same kind of thing.
11									line	Others-Cousin-State	what	It might be worth calling up Memorial Hospital's ED to ask about John Smith.
12					and	not		Memorial-hospital-ed	line	Ask-Patient-Spec	how	Click in the actors bar at the top-left of the screen on the icon for Memorial Hospital's ED and talk to someone there.
13											why	If John Smith has the same thing Ryan Smith has, then the folks at Memorial may have useful information, or by pooling what you've learned so far, you may get new insights into the case.
14											pro	You should probably consider following up with Memorial Hospital's ED on how Ryan's cousin is doing.

Figure 66. Behavior Spreadsheet with Complex Scenario-Specific Tutoring Line.

The main limitations on script line expression enforced by the spreadsheet format include (1) logic and control structures can only be nested two-deep, and (2) tutor actions are implicitly limited to being say operations combined by the `alt` operator. These turn out not to be meaningful limitations when reviewing the behaviors that had to be authored for our initial Anthrax scenario. In fact, logically speaking, it is always possible (thought perhaps not convenient) to express an arbitrarily nested logical expression in terms of two layers (e.g. an `and` containing `ors` or an `or` containing `ands`) plus negation.

10.4 Text-Editing for Authoring

Though we ended up with a spreadsheet format that is more expressive than we originally expected (and likely as expressive as will ever be needed), we do not necessarily expect all behavior authoring to be done using spreadsheets. Much of our authoring during this project was accomplished using COTS (free, open source) text editors. As developers we preferred to use GNU Emacs, a very high-end programmers' text editor that, though idiosyncratic, allows for endless customization and innately provides automatic parentheses-balancing and indentation. Other users would likely prefer a simpler text editor such as Notepad++, which hews more closely to Windows's editing conventions, but still provides a form of parentheses-balancing and indentation, plus open/close toggles for balanced/indented forms.

Figure 66 shows Notepad++ displaying the `spread_` file generated from the `patient.ods` spreadsheet (originally shown in Figure 65 and Figure 66). This is precisely the same human-readable format used in the `script_` files. In fact, if desired behaviors initially generated from spreadsheet data could be cut from this file and pasted into the corresponding `script_` file and then further elaborated, should that prove necessary. The most complex parts of such files are the expressions in the Interpreter language as described earlier in Sections 8.1 (the general classes of Enact operators) and 8.4 (the core language). Appendix E provides a more gradual introduction to the Enact scripting language, based on a set of examples drawn from an actual scenario script.

```

1  (require (lib "enact.ss" "its"))
2
3  (script (patient * *))
4
5
6  ;; =====
7  ;; Scenario-specific lines about cousin's illness
8
9  (-> Others-Cousin-State
10  (or (hear "How is your cousin?")
11      (hear "Do you know how your cousin is doing?")
12      (hear "Is your cousin as sick as you are?")
13      )
14      (say "My wife told me that his wife called to say he was heading over to the Memorial Hospital,
15      so I guess pretty bad.")
16      )
17
18  (-> Others-Cousin-Name
19  (or (hear "What is your cousin's name?")
20      (hear "Can you tell me your cousin's name?")
21      (hear "Who is the cousin you went to the game with?")
22      )
23      (say "My cousin's name is John Smith.")
24      )
25
26  (-> Others-Suggest-Call
27      #f
28      :point ED-Diagnosis-Infection
29      :rank 9
30      :when "00:10:00"
31      :test (and (or (line Complaint-Others)
32                    (line Others-Cousin-State)
33                    )
34              (and (not (as Memorial-hospital-ed (line Ask-Patient-Spec)))
35                  )
36              )
37      :hint (say "Given this patient's condition, it's interesting that other people he was with may
38      have the same kind of thing.")
39      :what (say "It might be worth calling up Memorial Hospital's ED to ask about John Smith.")
40      :how (say "Click in the actors bar at the top-left of the screen on the icon for Memorial
41      Hospital's ED and talk to someone there.")
42      :why (say "If John Smith has the same thing Ryan Smith has, then the folks at Memorial may
43      have useful information, or by pooling what you've learned so far, you may get new insights into
44      the case.")
45      :pro (say "You should probably consider following up with Memorial Hospital's ED on how
46      Ryan's cousin is doing.")
47      )
48  )

```

Figure 67. Generated spread_ File as Viewed in Notepad++ Text Editor.

11 Appendix E: Sample Agent Script Lines

This appendix contains selected script lines from the METTLE scenario presented in Appendix C, chosen to illustrate aspects of the system’s agent scripting capabilities. Each simulated character has a “script”¹¹ composed of a set of “lines.” Each line is a kind of rule, primarily composed of a “cue” and a “response.” When the cue happens, the simulated agent may generate the corresponding response. Cues are composed from a language of *tests*. Responses are composed from a language of *actions*.

Both the test and action languages are written as parenthesized lists, where the first element of the list specifies the operation (i.e. which test or action to perform) and the other elements flesh out the operation. For example, a common test is *hear*—written as the parenthesized list “(hear <string>)”, which means the agent should test whether it has heard the student say something like the whatever appears in the <string> position. A common action is *say*—written as the parenthesized list “(say <string>)”, which means the agent should output whatever appears in the <string> position.

11.1 Simple “What’s your name” Script Line

The following is an example of the simplest kind of script line designed to handle an interaction that might come up during the patient interview. Each script line is a parenthesized list that starts with either `->` or `=>`, followed by a **name**, a **cue**, and a **response**. The `->` means this is a rule that should be run at most once. The rule name—in this example ‘Patient-Name’—is handy for keeping track of (and testing) which rules have or have not yet been used during a scenario. In this case, the rule cue makes use of the *hear* test described above; it also makes use of the *or* test as a way of combining two alternative *hear* tests. The test language allows the standard Boolean connectors—*and*, *or*, and *not*—with their normal meanings. In this case, the cue will be considered satisfied if either of the two *hear* tests pass. Finally, the rule response uses the *say* action described above.

```
(-> Patient-Name
  (or (hear "What is your name?")
      (hear "What do they call you?")
  )
  (say "My name is Ryan Smith.")
)
```

The meaning of this rule is that if the simulated patient hears the student say either “What is your name?” or “What do you call yourself?” then the patient will respond with “My name is Ryan Smith.” The student does not have to type exactly either of the two expected *hear* strings for the system to recognize what they might be trying to say. The student just has to get close enough for the system to propose the match, and then they have to confirm their intended meaning. Also note that recorded speech can optionally be associated with this script line simply by placing an appropriate media file (e.g. a .wav sound file) in the appropriate directory (e.g. the character’s *speak* directory) with the same name as the script line (e.g. in this case *Patient-Name.wav*).

Finally, it is worth noting that METTLE allows for composition of scripts and even script lines from different sources. In particular, a set of default scripts can be defined that apply to classes of simulated characters. In the case of a patient, we have defined a basic set of several hundred script lines in a default file, providing a reusable set of rule names and cues covering many standard interview questions, examination actions, diagnostic tests, and so on. The result is that when scripting a particular scenario, the rule shown above could be written with the cue left blank, to be filled in from the default definition.

¹¹ Actually each simulated agent may have a *set* of scripts, with different scripts being active in different *scenes* of the larger scenario, or as the agent’s *state* is modified.

11.2 Asking About the Chief Complaint (With Tutor Commentary)

This second example is a more complex patient interview script line. It is also a run-once rule (->), this time named 'Complaint'. The #f indicates that the cue is being left out, and will be provided by a default file; that default provides a cue built from the or of a bunch of hear tests (several different ways of eliciting a chief complaint). The response in this case is a compound action, designated by the seq form which specifies that its arguments are a set of actions to be executed sequentially. Those actions are a say action (as we've seen before), and a post action (which posts important findings to a designated section of the patient chart—here the complaint page).

The important novel aspects of this rule, however are the pieces that come after the response—the fields labeled :test, :rank, :hint, :what, :how, and :why—which all specify aspects of Tutor behavior related to this patient behavior. The :hint, :what, :how, and :why fields are very simple. Each of them contains an action specification (in the same action language as is used to specify the main rule response) that is to be executed when the student clicks on the 'Hint' 'What?' 'How?' and 'Why?' buttons on the Tutor side of the Player screen. We call these four fields collectively the Tutor hints. Here (as in all the examples in this scenario), if the script line's Tutor hints are active when the student clicks one of those buttons, then the Tutor will say whatever appears in the corresponding say form.

The :test field determines when this script line's set of Tutor hints might be active. The :test field contains an expression in the test language. Here we see a Boolean operator (not) and a new test type: line. The line test checks to see whether the script line with the given name has ever run. Given the not, the test here checks to see if the named line has *never* run. The script line being checked in this case is *this* script line. So effectively, the test is saying: "If this patient behavior has never been triggered, then make the Tutor hints associated with this behavior active." The :rank field is used to assign a priority, so if a bunch of Tutor hints are active, the most important ones will be offered first.

```
(-> Complaint
#f
(seq (say "I feel lousy, I've been coughing, I've been sweating
        at night. I've had chills, headache, I've thrown up a
        couple of times, my stomach hurts and sometimes my
        chest hurts too.")
      (post complaint
        "Patient complains of coughs, night sweats, chills,
        headache, stomach and chest pain, and vomiting.")
    )
:test (not (line Complaint))
:rank 10
:hint (say "You might want to interview the patient.")
:what (say "In this program you can carry out a simulated patient
        interview to learn about their condition in their own
        words.")
:how (say "Try typing into the box on the patient's side of the
        screen labeled 'Type what you want to say, then click
        Try'. You might start by asking about the chief
        complaint. (Don't forget to click 'Try' once you've
        typed your question.)")
:why (say "It's generally a good idea to get what information you
        can directly from the patient.")
)
```

11.3 A Curriculum-Relevant Behavior with Proactive Tutor Prompting

This third script line example presents another patient interview behavior, but this time one that directly relates to the training's underlying curriculum, and one that is set up to proactively prompt the student to take an action (if they have not taken it within some specified time). As with the previous example, the cue for this rule is provided in the patient default behaviors file; in this case, it looks for any of several ways to ask if the patient knows anyone else who seems to have the same kind of illness that they are suffering from. The response highlights a piece of information that was actually included in the briefing: that our patient has a cousin John who has a similar illness.

The important novel aspects of this rule are the appearance of several new Tutor-related fields: `:pro`, `:when` and `:point`. The `:pro` field is similar to the `:hint`, `:what`, `:how`, and `:why` fields in that it contains an action to be executed by the Tutor. The difference is that the `:pro` action will be executed *proactively*—that is, on the Tutor's own initiative, and not in response to a student button click or other action. Such a `:pro` action is executed if the containing line is not triggered within a specified amount of time after the line's `:test` becomes true (or if, as in this case, there is no `:test`, then after the start of the scenario); the time limit is specified in the `:when` field. The effect here, then, is to prompt students to ask if the patient knows any other similarly sick people, if they haven't gotten around to asking within 5 minutes of the start of the session.

Finally the `:point` field links this behavior to a node in the system's curriculum tree; each such node represents something (or a set of things) that students are expected to know or be able to do. The linkage of a point to a script line in this way means that the triggering, or failure to trigger, the line gives evidence about the student's state of knowledge and skill. If the student takes the triggering action at the appropriate time—when the `:test` is true (or absent), and before the `:when` limit has expired—then they get credit for good performance on the linked point. If they take such action at the wrong time—when the `:test` is false, or too long after the `:test` becomes true—then they lose credit. Likewise, if they have to ask for hints, they lose credit on associated points.

```
(-> Complaint-Others
  #f
  (say "Yeah, my cousin John has come down with some fluey thing
        since we last saw each other. My wife says his wife took
        him to Memorial Hospital today.")
  :pro (say "You might consider asking whether Ryan knows anyone
            else who has what he has.")
  :when "00:05:00"
  :point ED-Diagnosis-Infection
)
```

11.4 A Generalized Cue with Negative Tutor Feedback

This fourth script line is the first whose cue is anything other than a disjunction of hear tests. This rule responds not to questions the student asks of the patient, but to choices made on an order form. The `choose test` compares selections from such a form against an underlying conceptual taxonomy. Here, the chosen items are to be drawn from a defined space of *treatments*. The interesting point is that since such spaces are hierarchically structured, it is possible to write general tests that cover many specific choices. The condition tested here—(`choose treatments:administer-antibiotics`)—covers any particular kind of antibiotic the student might order.

Note that this is the first script line we have seen that contains no response. This reflects the level of simulation targeted in this scenario. We do not expect the simulation to run long enough to see any particular result (change in patient state) from giving the patient antibiotics. Instead, the point of this line

is to provide a place to attach Tutor behaviors. As in earlier examples, the `:test` field indicates when this behavior starts to become appropriate—in this case, after the student has ordered a blood culture (that is, after they have triggered the script line `Test-Micro-BC`, which is the response to ordering such a test). The new Tutor behavior field introduced here is the `:nope` field, which contains a Tutor action to be executed if the student triggers this script line at a time when the `:test` is not true. The effect is that if the student treats the patient with any antibiotic before they order blood cultures, the Tutor speaks up to deliver negative feedback explaining why this is not a good idea. Note also that with the `:point` field filled as it is, the student model will also be updated with notes about the appropriate or inappropriate sequencing of activity, depending on whether the line is triggered when the `:test` is true or false.

Finally, note that this is our first example of a `=>` rule which can be triggered any number of times. We chose this form because more than one course of antibiotic treatment can reasonably be ordered. If we wanted to avoid the repetition of the Tutor's negative feedback, we could convert this to a run-once rule. Alternately, we could change the form of the `:nope` action to use the `alt` operator as a way of specifying alternate responses to be used on successive occasions of the action being executed; once all the alternatives have been run through once, the last alternative is repeated as many additional times as needed, and if necessary, that last alternative can be a “no-op”—that is, the suppression of further Tutor response.

```
(=> treatments:administer-antibiotics
  (choose treatments:administer-antibiotics)
  :test (line Test-Micro-BC)
  :nope (say "Although there is some controversy, in a patient as
            sick as this one it is considered inappropriate to
            begin administration of antibiotics before collecting
            blood for culture.")
  :point ED-Sequencing-BloodSamplesBeforeAntibiotics
)
```

11.5 A Complex Test with Positive Tutor Feedback

This fifth example introduces the `:yup` field—the opposite of the `:nope` field described above. It is used to define *positive* feedback to be delivered by the Tutor—that is, Tutor behavior to be executed when the student triggers the script line when the `:test` is true (remember, `:nope` specifies a Tutor behavior to execute if the `:test` is false). Here we see a `:yup` combined with a set of hints. In theory a single script line could carry any combination of hints, proactive prompting, and both positive and negative feedback. In practice, we have found it is better to be more sparing, and to create variety by choosing Tutor behaviors to suit the importance of the issues in question, and expectations about what students are likely to do. Tutor behaviors can also be chosen to try to ensure continued student progress and overall scenario flow.

This example is also interesting because it has the most complex `:test` we have seen so far. This test is a Boolean `and` combination of three more specific `line` tests (two of those wrapped in the `not` negation form). Effectively what is going on here, is (1) the first `line` test is establishing that an action which is considered an important one to do earlier than getting X-Rays has been taken care of; then (2) the second (negated) `line` test is checking that this X-Ray action has not already been done (since it is an action that can be done multiple times), and (3) the final (negated) `line` test is checking that an equivalent action has not already been done (the ordering of a chest CT).

```
(=> Test-Image-XRay-Chest
  #f
  :test (and (line treatments:establish-IV)
             (not (line Test-Image-XRay-Chest))
             (not (line Test-Image-CT-Chest))
           )
  :rank 8
  :hint (say "Given a complaint about shortness of breath you might
            want to get imagery that could shed light on the
            underlying problem.")
  :what (say "Consider ordering X-Ray imagery of the patient's
            chest.")
  :how (say "Click the 'Test' button and choose the 'Imagery
            Studies' menu item, then order the Chest X-Ray.")
  :why (say "Findings suggestive of pulmonary problems can be
            investigated using a chest X-Ray to image the lungs
            and surrounding structures.")
  :yup (say "Yes, a Chest X-Ray makes sense as a way of seeing what
            is going on with the patient's shortness of breath.")
  :point ED-Diagnosis-ShortnessOfBreath
)
```

11.6 A Pure Control Script Line

The last script line we consider here is an example of a pure scenario control rule. This rule's cue looks for conditions that signal the end of the scenario, and its response initiates the Tutor's scenario wrap-up dialog. The cue and response introduce a couple of additional important operators that work across both the test and action languages: the as operator allows a nested expression to be interpreted with respect to some named agent, and the dialog operator allows for testing and execution of more complex agent-initiative interactions than can easily be scripted by use of normal *hear/say* script lines.

What this rule does is first to check if the tutor has run either of a pair of dialogs that are designed as penultimate steps in any scenario run. The *diagnosis-sterile-results* and *diagnosis-enough-results* dialogs are both Tutor-initiative dialogs that explore the student's thinking about possible diagnoses; the dialogs differ in that they are appropriate to two different states of student knowledge—that is they are chosen based on different constellations of tests the student might have run. It then checks whether another important dialog has been run, which has the student interacting with the simulated hospital administrator in a discussion of possible responses to an anthrax diagnosis. Finally it checks that the Tutor's scenario wrap-up dialog has not already been run. If all those conditions are met, then the rule launches the Tutor's wrap-up dialog.

```
(=> Ending:Wrap
  (and (as tutor (or (dialog run? diagnosis-sterile-results)
                    (dialog run? diagnosis-enough-results)
                  ))
       (as hospital-administrator (dialog run? choices))
       (not (as tutor (dialog start wrap))))
  )
  (as tutor (dialog start wrap))
)
```

12 Appendix F: Sample Tutor Dialog

METTLE is designed to support two different modes of conversational interaction with simulated characters: student-initiative and agent-initiative. The script lines described in Appendix E are primarily designed to support agent response to *student-initiative* conversation. The dialog structures presented in this appendix control *agent-initiative* conversation. The major structures to be discussed are **dialogs**, **topics**, **points**, **queries**, and **responses**.

Each dialog is defined in its own file, and the name of that file is used in script line tests and actions to query whether the dialog has run, and to start or stop it (as exemplified in B.6 above). Like most other METTLE data structures, dialogs are written as parenthesized lists, nested inside one another, where each list is headed by a flag that says what kind of object is being defined, followed by a name for that object, and then a bunch of keyword/value pairs designating the fields of those objects. Each dialog contains a single dialog form that looks something like the following example:

```
(dialog diagnosis-enough-results
  :intro (say "I'm interested in where you've gotten to in your
            diagnostic thinking.")
  :topics (<topic list>)
  :extro (say "Based on this review, you should feel reasonably
            confident that you have a handle on what is most
            likely going on with this patient.")
)
```

This is the top-level of a Tutor dialog named 'diagnosis-enough-results' intended to explore and clarify the student's diagnostic thinking at a point in the interaction where we know the student has uncovered enough evidence that they should be able to settle on a correct diagnosis. The :intro and :extro fields (which we will see recurring across most of the nested dialog structures) contain actions of the same sort that we saw used earlier in script lines (here we see say actions). The :intro action is done at the start of the dialog; the :extro action is done at the end of the dialog. In between, the bulk of the dialog is managed by processing through as many of the topics from the <topic list> as become active along the way.

In the following sections we look at some representative topics to see, (1) how they become active, (2) how they are processed, and in particular, (3) how their internal structures are defined and used.

12.1 A First Simple Unconditional Topic

Each topic, in addition to its required name, and its optional :intro and :extro actions, has a :test that controls when or whether it is introduced into the running dialog. When that test is satisfied, the topic gets put on a queue. When a topic comes to the front of the queue (as other earlier activated are exhausted), it is processed much like a sub-dialog: its :intro action is done, its main body is processed, and finally its :extro action is done. Where the main body of a dialog is a list of topics, the main body of a topic is a list of points. We think of a topic as something a simulated agent might want to talk about during a dialog. We think of a point as something (within a topic) about which the agent might want to get evidence from the student; a point is something a student should demonstrate they “get” or optionally become convinced of by argumentation.

The example in this section is a topic called 'Topic:Diagnosis'. Its :test is the simple form (true) which is a test operator that always succeeds, thus this topic is always active; since it sits at the beginning of the dialog's topic list it will always be the first topic onto the topic queue, and thus the first topic that gets processed. This topic has no :intro nor :extro actions. It has a single point named

'Point:Diagnosis-Ready'. While we will see more interesting examples of points in later sections, this simple point has enough substructure to gain some idea of how points are processed.

The main body of a point is a list of queries. If a point is something about which the agent wants to gauge or induce user understanding, a query is an opportunity for interaction with the user to test or nudge their understanding. Beyond the normal (and optional) :intro and :extro actions, the processing of a point is largely the processing of its queries. Normally a point has a :test condition, and later queries will be ignored as soon as that :test is satisfied (that is, as soon as a user response to one of the queries demonstrates that the user got the point). The processing of each query is similar to other objects we've seen along the way—start with the :intro action, do something, then end with the :extro action—but in this case, the *do something* involves stopping to accept input from the user.

The single query we have in this example is named 'Query:Diagnosis-Ready?', and its :intro action poses a question to the user “*Do you think you have enough information on hand to settle on a diagnosis?*” The field :pick-ordered whose value is true is a flag that specifies (1) this query should be handled as a multiple-choice question rather than as free-text type-in, and (2) that the choices should be presented in a fixed rather than a random order. The choices that will be available are defined in the query's list of responses. To complete the picture, each response is an object with :test, :intro, and :extro. The :test captures the recognition conditions to determine if the user has generated a particular response to a particular query; here one response represents an answer of “Yes” and the other an answer of “No” to the query about whether being ready to form a diagnosis. Depending on what the user picks as their input, responses may be recognized and their actions performed.

```
(topic Topic:Diagnosis
  :test (true)
  :points
  ((point Point:Diagnosis-Ready
    :queries
    ((query Query:Diagnosis-Ready?
      :pick-ordered true
      :intro (say "Do you think you have enough information on
                  hand to settle on a diagnosis?")
      :responses
      ((response Response:Diagnosis-Ready?-No
        :test (hear "No.")
        :intro (say "Actually, you've got most of the key
                    information you need to make a good case
                    for a diagnosis.")
        )
      (response Response:Diagnosis-Ready?-Yes
        :test (hear "Yes.")
        :intro (say "Yes, you've got most of the key
                    information you need to make a good case
                    for a diagnosis.")
        )
      )
    )
  )
)
```

In sum, the effect of this topic is just to ask the student a yes/no question about whether they think they are ready to reach a diagnosis, and to provide some feedback on their answer that should orient them to the ensuing discussion of what that diagnosis might be.

12.2 A Topic with Repeated Probes and Reference to Concept Spaces

The second topic we will look at is the next topic in the same dialog. It too has a `(true) :test` and lacks both `:intro` and `:extro`. It also has just a single point, and that point has no `:test` of its own. The difference is that this topic's single point has a succession of three queries, which are repeated probes, giving the student three chances to give essentially the same information—"What conditions are in your differential now..."—though with increasing levels of prompting and specificity.

The new features introduced here are (1) the use of a `:test` on a query, and (2) the use of a `:concepts` list in place of explicitly enumerated `:responses`; these are related, since the test forms used here check whether the user has made certain kinds of choices from a concept space. The concept space in question here is the system's defined taxonomy of conditions (diseases). Since METTLE is focused on teaching about CBR, its concept space for conditions is, at the highest level, divided up into (a) chemical, (b) biological, (c) radiological, and (d) conventional conditions. Biological agents are in turn broken down into bacterial-agents, viral-agents, and biological-toxins.

When a query is specified with a `:concepts` list, that means the expected responses to the query are to be found in any of the concept trees in that list. The first two queries here specify the same value: `conditions:root` which means any item anywhere in the conditions tree. The third query specifies two more specific values: `conditions:bacterial-agent` and `conditions:conventional-condition` which means only condition concepts in those two sub-trees will be considered. This makes sense, because the third query asks more specifically about "...serious bacterial infections." Note that the use of the `:pick-ordered` flag in these three queries means that the program will prompt the user with a tree of checkbox items mirroring the (specified pieces of) the defined concept space.

Finally, if we look at the `:test` conditions on the queries we see that (1) the first query is always asked (it has no `:test`); (2) the second query is asked if the student did not choose any bacterial-agents or conventional-conditions in answer to the first query; and (3) the third query is asked if the student did not choose anthrax in particular as an answer to either of the previous two queries. Again, remember that this dialog is written specifically for a point in the scenario where the student is known to have gathered enough evidence to strongly suggest anthrax is the correct diagnosis.

```
(topic Topic:Differential
  :test (true)
  :points
  ((point Point:Differential-Items
    :intro (say "Let's talk about the set of conditions you are
              currently considering.")
    :queries
    ((query Query:Differential-Items1?
      :intro (say "What conditions are in your differential now,
                    based on the patient's original complaint,
                    plus the information you've gathered yourself
                    through simulated interviews, examinations,
                    and tests?")
      :pick-ordered true
      :concepts (conditions:root)
    )
  )
)
```

```

(query Query:Differential-Items2?
  :test (or (not (choose = conditions:conventional-condition))
            (not (choose = conditions:bacterial-agent))
          )
  :intro (say "Is there anything else you think you should be
              considering at this point?")
  :pick-ordered true
  :concepts (conditions:root)
)
(query Query:Differential-Items3?
  :test (not (choose = conditions:anthrax))
  :intro (say "Consider other possibilities from the universe
              of serious bacterial infections.")
  :pick-ordered true
  :concepts (conditions:bacterial-agent
             conditions:conventional-condition)
)
)
)
)
)
)

```

12.3 A Topic with Nested Point Structure

While the dialog we have been sampling contains many more topics (e.g. discussions of the evidence against a wide range of lower quality diagnostic hypotheses), this final example topic is probably the most complex that has been authored for the current METTLE scenario: it reviews the key evidence in favor of a diagnosis of inhalation anthrax for this patient. Since it is rather long, we break it up into pieces by pulling out key points, just as we have been breaking up our discussion of the overall dialog by pulling out key topics. Here is the outline of the topic:

```

(topic Topic:Anthrax
  :test (or (choose = conditions:anthrax)
            (node Query:Differential-Items3?)
          )
  :intro (say "Anthrax, in particular, inhalation anthrax, should
              be at the top of your list of diagnoses at this
              point.")
  :points (<list of points>)
  :extro (say "With the evidence in hand we should certainly be
              considering anthrax as a possible or even likely
              diagnosis. Given the seriousness of the disease
              for this patient, and potentially for the community
              we should probably:<ol>
                <li>Start aggressive treatment</li>
                <li>Seek an informed second opinion and further
                    confirmation</li>
                <li>Start getting the word out to engage a larger
                    scale emergency response.</li></ol>")
  )
)

```

The only points to note about the top level of this topic are the `:test` and the formatting in the `:extro`. The `:test` is interesting because the effect we are trying to achieve here is to make sure this topic is discussed, *whether or not the student actually suggests anthrax as a likely diagnosis*. While this could be achieved simply by using a `(true)` test, that would have the slightly problematic effect of putting this topic on the queue earlier than other topics that get triggered by student answers to queries about what is in their differential. The test as written, triggers this topic, either when the student explicitly says they are considering anthrax, or if the Tutor gets around to using its third and final differential probe query (which only happens if the student has not yet mentioned anthrax). The `:extro` is interesting because it demonstrates that we can use standard HTML formatting when writing agent output: the `...` tags are HTML format markers that produce a bulleted list in the Tutor's output.

The `Topic:Anthrax` that we are looking at here breaks its discussion of the evidence in favor of anthrax up into three major points: (1) the evidence from the initial patient presentation, complaint, interview, and examination, (2) the evidence from tests, and (3) the evidence from imagery. We will focus our attention on the point that deals with test results, specifically with microbiological tests that reveal features of the infectious organism (we ignore the other points as repetitious for this exposition). The name of the point in question is `'Point:Anthrax-Tests'`, and though it has no queries of its own, it has two nested points: `'Point:Anthrax-Tests-Infection'`, and `'Point:Anthrax-Tests-Bacteria'`. For simplicity's sake, the first nested point is treated more like a minimal topic—it simply executes an `:intro say` action. The second nested point is more interesting, and better fits the intended use pattern for points.

That more interesting nested point, `'Point:Anthrax-Tests-Bacteria'` has a `:test` that is looking for some particular things the student should say. In response to queries asking which findings about the bacterial infection are most important, it wants the student to observe that findings of (1) rod-shaped bacteria and (2) gram positive bacteria are together critical to diagnosing anthrax. A point that carries a `:test` like this will also typically carry two alternative fields that function much like the normal `:extro` field. The `:success` action will be done if the student gives input that satisfies the point's `:test`; the `:failure` action will be done if the student does not give input that satisfies the point's `:test`.

This point is also structured with three queries (chances for the student to exhibit their correct understanding, or to lead them to that understanding). The first is a relatively broad question prompting for any findings about the bacterial infection. The second and third queries are more directed questions asking specifically about bacterial morphology and gram-stain; they are only used in the case where the student has not satisfied the entire point by their answer to the first query (else the point processing would immediately terminate with execution of the `:success` action), and have not provided the relevant part of the desired answer (else these queries' individual `:tests` would not be satisfied).

```

(point Point:Anthrax-Tests
  :intro (say "Now let's review the evidence from the tests you
              ordered for this patient.")
  :queries (#f)
  :points
  ((point Point:Anthrax-Tests-Infection
    :intro (say "The key tests are those that established the
                presence of an infection.  You could find such
                results for blood, sputum, bronchoscopy
                washings, pleural tap, and CSF.  This patient
                is suffering from a major infection!")
    )
  (point Point:Anthrax-Tests-Bacteria
    :test      (and (choose = findings:rod-bacteria)
                   (choose = findings:gram-positive-bacteria)
                 )
    :success (say "Right.  Arriving at a correct diagnosis of
                  anthrax is aided tremendously by recognizing
                  that in this context gram-positive bacilli
                  are pretty distinctive.")
    :failure (say "Actually, what should start to ring warning
                  bells is the presence of a serious gram
                  positive bacilli infection in the context of
                  these severe flu-like symptoms.")
    :queries
    ((query Query:Anthrax-Tests-Bacteria1?
      :intro (say "What attributes of the infection are
                  particularly distinctive and worrying?")
      :pick-ordered true
      :concepts      (findings:infection-finding)
      )
    (query Query:Anthrax-Tests-Bacteria2?
      :test (not (choose = findings:rod-bacteria))
      :intro (say "What about the bacteria's morphology?")
      :pick-ordered true
      :concepts      (findings:bacteria-shape)
      )
    (query Query:Anthrax-Tests-Bacteria3?
      :test (not (choose = findings:gram-positive-bacteria))
      :intro (say "What about the bacteria's gram-stain?")
      :pick-ordered true
      :concepts      (findings:bacteria-gram-stain)
      )
    )
  )
)
)
)

```

13 Appendix G: METTLE Help Pages Explaining System Use

13.1 METTLE On-Line Help Index

[Next](#)

METTLE (*the Medical Emergency Team Tutored Learning Environment*) is a computer-based role-play training system for medical professionals who want simulated practice coping with Chemical, Biological, and Radiological (CBR) emergency situations.

The basic sequence for using METTLE is as follows:

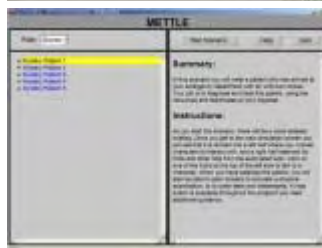
1. Launch METTLE:



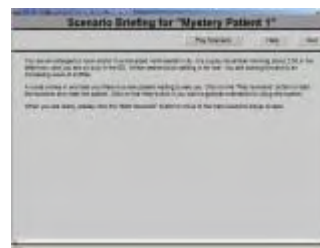
2. Log In:



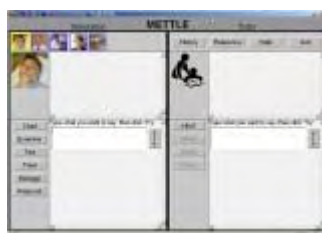
3. Choose a Scenario:



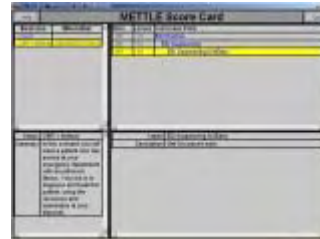
4. Get Briefing:



5. Play the Scenario:



6. Review your Play:



The following more detailed help pages are available:

1. Launching METTLE	Help with the Launch Screen
2. Logging-In to METTLE	Help with the Login Screen
3. Choosing a Scenario	Help with the Scenario Chooser Screen
4. Getting Briefings	Help with the Briefing Screen
5. Playing a Scenario	Help with the Scenario Player Screen
Conversing with Character	Help with how to converse with a character
Accessing Patient Charts	Help with simulated patient charts
Examining a Patient	Help with the patient examination
Ordering Patient Tests	Help with ordering patient tests
Ordering Patient Treatments	Help with ordering patient treatments
Managing Patient Care	Help with managing other aspects of patient care
Responding to an Emergency	Help with responding to a systemic emergency
6. Getting Scenario Scorecard	Help with the after-scenario Report Card screen
Reviewing Scenario History	Help with the Transcript windows
Reviewing Reference Materials	Help with the Reference Materials window
Getting Help	Help with the Help screens


[Next](#)

13.2 Launching METTLE

[Previous](#)

[Index](#)

[Next](#)

METTLE is a web-based application. That means there must be a METTLE server running somewhere that you can access with your web browser. If you are loading this help page from a server, chances are it is also running METTLE. If so, you should be able to launch METTLE by clicking this button: 



[Previous](#)

[Index](#)

[Next](#)

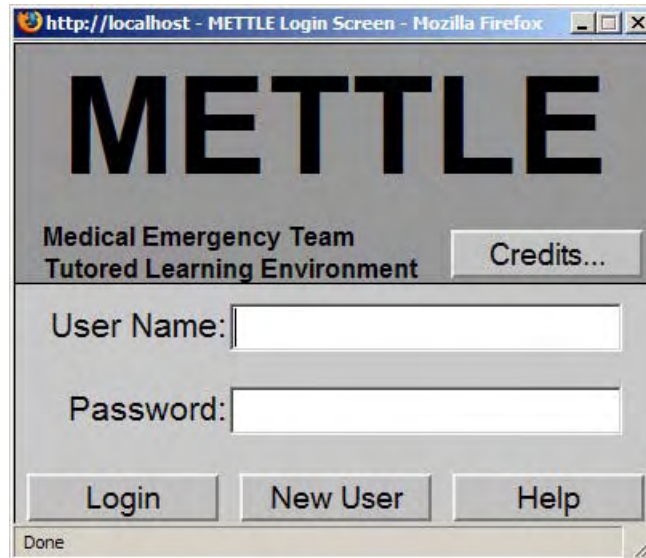
13.3 Logging-In to METTLE

[Previous](#)

[Index](#)

[Next](#)

When launched METTLE will display the Login screen shown below (although not necessarily in such a nicely sized window). This screen includes fields and buttons that allow you to identify yourself to the system so it can retrieve your student records. If you are new to METTLE, you can create a new student account for yourself.

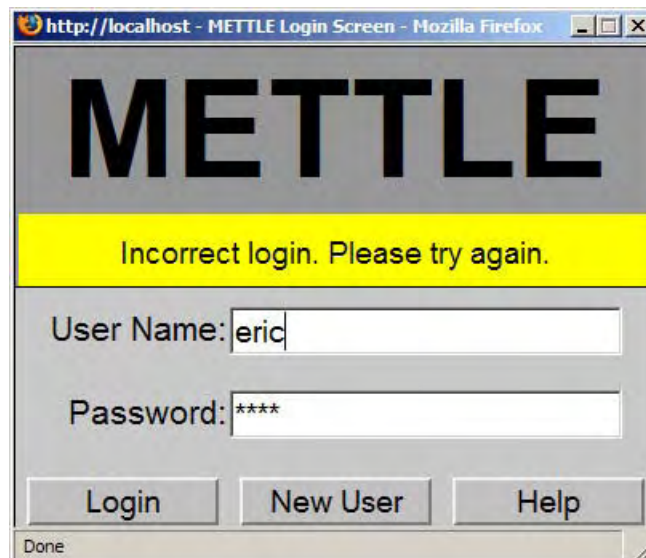


The screenshot shows a web browser window titled "http://localhost - METTLE Login Screen - Mozilla Firefox". The main heading is "METTLE" in large, bold, black letters. Below it, the text "Medical Emergency Team" and "Tutored Learning Environment" is displayed. To the right of this text is a button labeled "Credits...". Below the text are two input fields: "User Name:" and "Password:". At the bottom of the form are three buttons: "Login", "New User", and "Help". A status bar at the very bottom shows the word "Done".

- **If you already have an account:**

Type your user-name and password into the fields provided on the Login screen. Then click the **"Login"** button. If you have typed a correct user-name/password pair, the system will take you to the [Scenario Chooser screen](#).

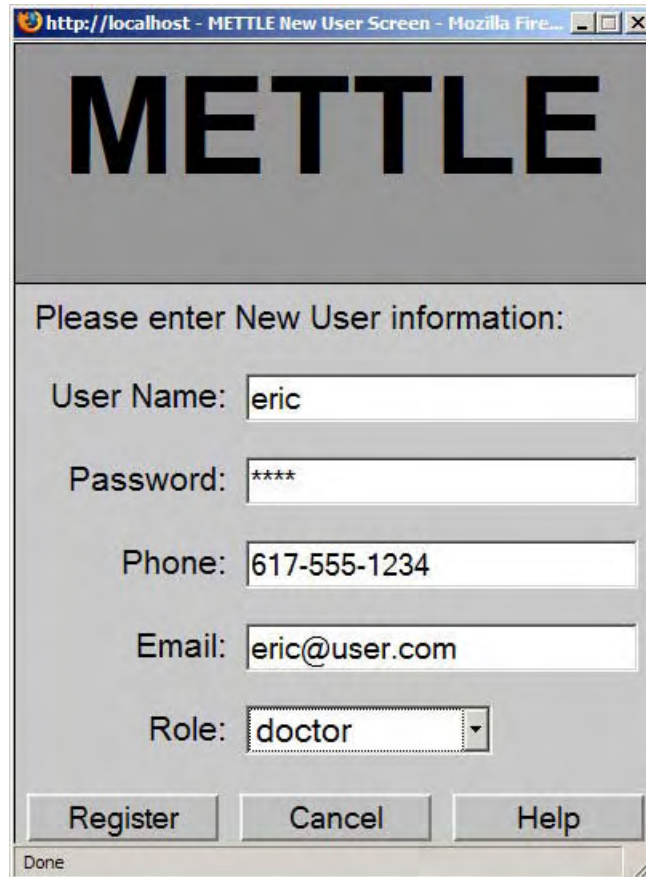
If your user-name or password are incorrect, you will get an error message, and be left at the Login screen.



This screenshot shows the same METTLE Login Screen as the previous one, but with an error message. A yellow banner across the middle of the form reads "Incorrect login. Please try again." The "User Name:" field now contains the text "eric" and the "Password:" field contains "****". The "Login", "New User", and "Help" buttons remain at the bottom. The status bar still shows "Done".

- **If you do not have an account:**

Click the "**New User**" button on the Login screen. That will bring up a dialog in which you can fill in basic information about yourself to establish your account. Clicking the "**Register**" button on this dialog will attempt to create the new account, and will take you back to the Login screen with the new user-name and password already filled in.



The screenshot shows a web browser window titled "http://localhost - METTLE New User Screen - Mozilla Fire...". The main heading is "METTLE" in large, bold, black letters. Below the heading, the text "Please enter New User information:" is displayed. The form contains five input fields: "User Name:" with the value "eric", "Password:" with the value "****", "Phone:" with the value "617-555-1234", "Email:" with the value "eric@user.com", and "Role:" with a dropdown menu showing "doctor". At the bottom of the form are three buttons: "Register", "Cancel", and "Help". A "Done" button is visible in the bottom-left corner of the browser window.

If you attempt to create a new account for a user-name that is already in use, you will get an error message, and be left at the New User dialog.

http://localhost - METTLE New User Screen - Mozilla Fire...

METTLE

User already exists. Please try again.

Please enter New User information:

User Name:

Password:

Phone:

Email:

Role:

Done

- **If you have forgotten your user-name or password:**

METTLE does not yet have any facilities to deal with this situation. Please just create a new account for yourself under a new name.

- **If you need help:**

These screens all have "**Help**" buttons that will bring you back to this page for a refresher on how to log in or create your account.

[Previous](#)

[Index](#)

[Next](#)

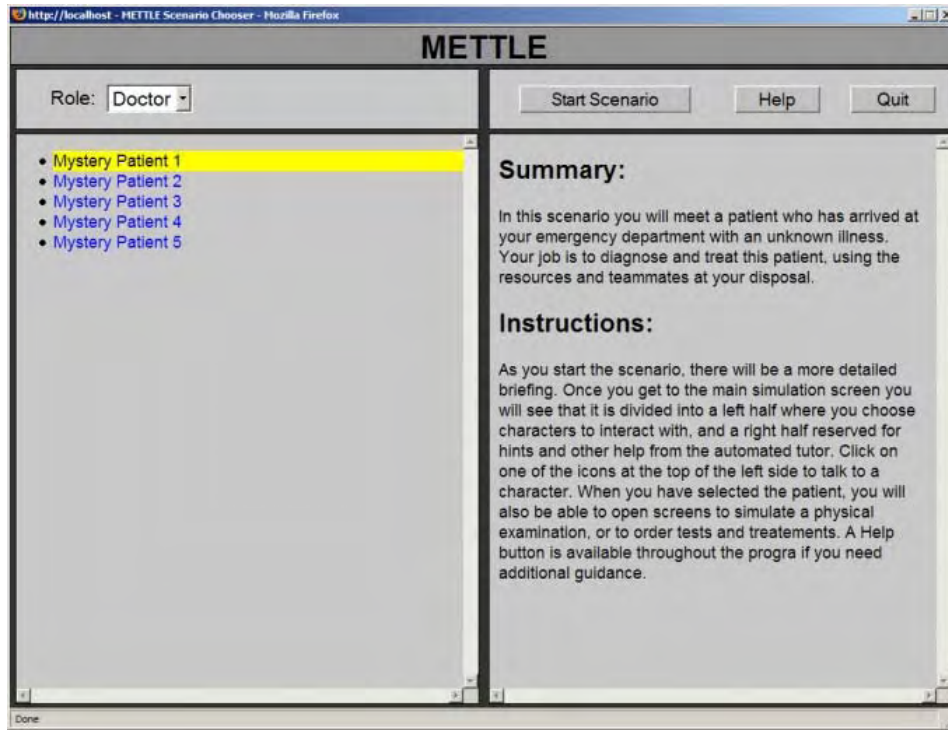
13.4 Choosing a METTLE Scenario

[Previous](#)

[Index](#)

[Next](#)

Once you have successfully logged into METTLE you will be taken to the Scenario Chooser Screen. For the time being, this screen is not terribly useful, as you will find that there is only one scenario currently available when you run the system.



- **Getting a List of Scenarios:**
The left half of the screen offers a view of the scenario library. It lists all of the scenarios currently available for you to choose among.
- **Viewing a Scenario Summary:**
Clicking on any of the listed scenarios causes display of scenario overview information in the right half of the screen. This includes a short summary of the scenario, as well as some general instructions.
- **Launching a Scenario:**
With a scenario selected as above, clicking the "**Start Scenario**" button loads the chosen scenario and opens the Scenario Briefing window.
- **Getting Help:**
Clicking on the "**Help**" button will bring up this screen.
- **Quitting METTLE:**
Clicking on the "**Quit**" button will close the Chooser Screen.

[Previous](#)

[Index](#)

[Next](#)

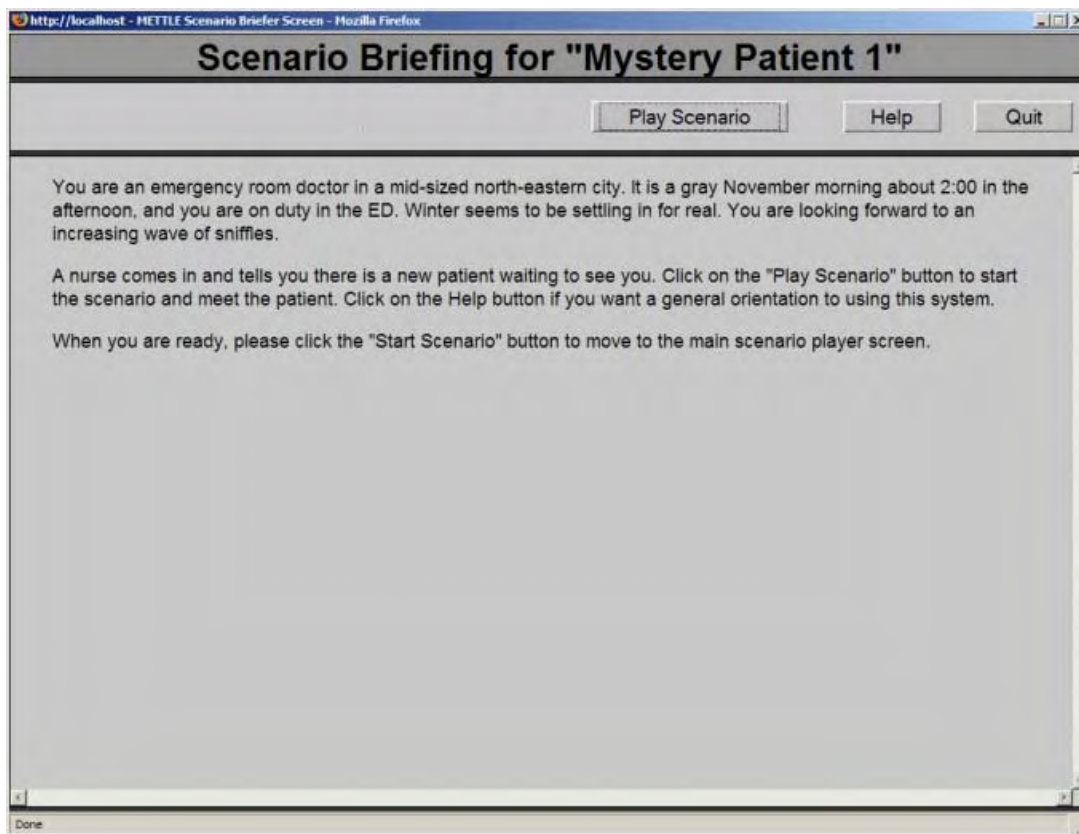
13.5 Getting a METTLE Briefing

[Previous](#)

[Index](#)

[Next](#)

At the start of each METTLE scenario, you will be brought to the Briefing Screen, shown below. The material on this display is intended to orient you to the situation and your task in the ensuing scenario. Note that some scenarios may have multiple *scenes*, and in that case there may be an additional briefing at the start of each scene as well.



- **Playing the Scenario:**

Once you feel comfortable with the briefing material, you can click the **"Play Scenario"** button to open the actual [Scenario Player](#) window and start the exercise.

- **Getting Help:**

Clicking on the **"Help"** button will bring up this screen.

- **Quitting METTLE:**

Clicking on the **"Quit"** button will close the Briefing Screen and return you to the [Chooser Screen](#).

[Previous](#)

[Index](#)

[Next](#)

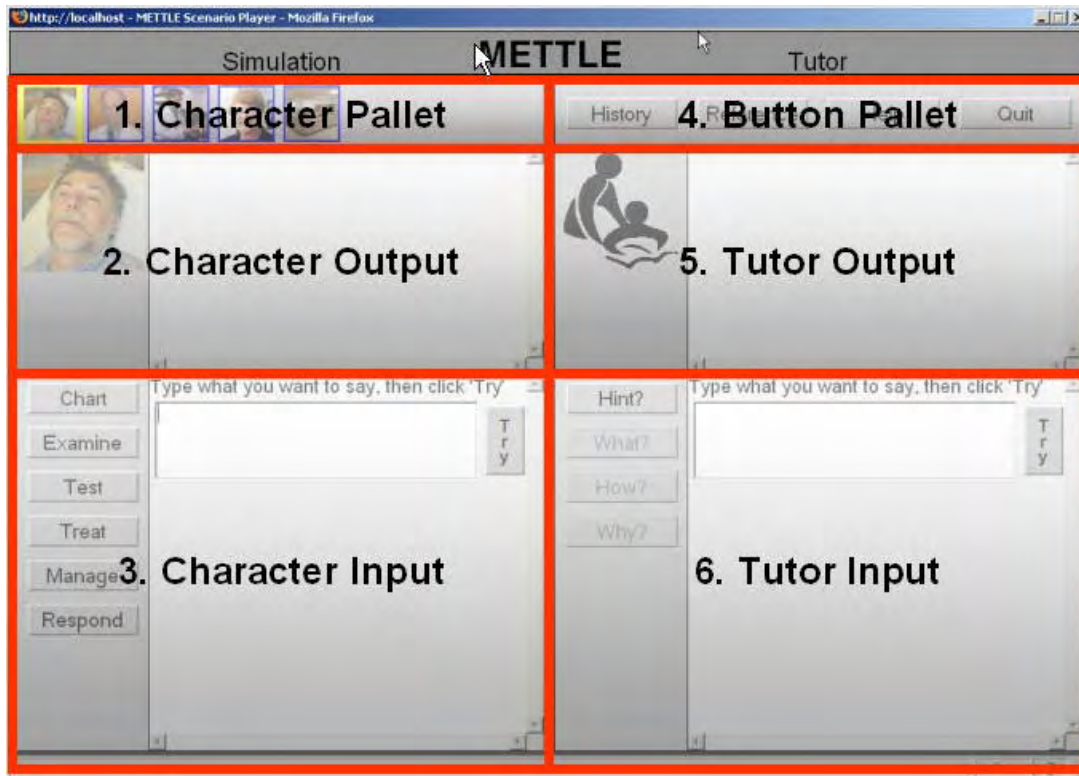
13.6 Playing a METTLE Scenario

[Previous](#)

[Index](#)

[Next](#)

The METTLE Scenario Player is where you will spend most of your time. The six sections of the screen layout are shown and discussed below.



1. **Character Pallet:** Contains an icon for each simulated character in the scenario. Clicking on an icon selects a character for input and output in the areas below. Grayed-out icons indicate characters that have nothing to say at the current time.
2. **Character Output:** Displays anything the currently selected character says (as well as what you say to them). Clicking the character's picture will also pop up a Transcript Window with a record of all your interaction with the character.
3. **Character Input:** Area where you can say things to the currently selected character:
 - To [converse](#) with a character, you can type in the text-box, and then click on the "**Try**" button to see how the system interprets your input. It will respond with a list of different possible meanings, or remain blank if the system could not find any interpretation. Click the check-boxes next to the meanings that capture what you were trying to say. If your intended meaning is not on the list, you can edit your text and click "**Try**" again.
 - For some characters, there may be additional buttons in the left margin. For *Patients* (as in the screen shot above) there will generally be buttons labeled "[Chart](#)", "[Examine](#)", "[Test](#)", "[Treat](#)", "[Manage](#)", and "[Respond](#)". These buttons (or pop-up menus they reveal) will generally open new windows to support specific kinds of interactions.
4. **Button Pallet:** Contains a basic set of buttons needed throughout the scenario:

- **History:** Pops up a [Transcript Window](#), much like the result of clicking on a character's picture, however the History transcript contains a complete record of all your interactions with *all* the characters.
 - **Reference:** Pops up a [Reference Library](#) window providing access to information that may be useful in working through the scenario, or following up on issues that came up during play.
 - **Help:** Pops up this [help page](#).
 - **Quit:** Closes the Player window and returns you to the [Chooser Screen](#).
5. **Tutor Output:** Parallels the *Character Output* area, but is reserved for the simulated Tutor. The Tutor may say things in response to your actions in the *Tutor Input* area below, or based on observations of your actions elsewhere in the system.
6. **Tutor Input:** Parallels the *Character Input* area, but is reserved for the simulated Tutor. It also provides a set of buttons for a common sequence of questions you might want to ask of the Tutor:
- **Hint?** Asks the Tutor to give a hint about what would be a reasonable next action to take.
 - **What?** Asks the Tutor to give a more directive hint about what to do next.
 - **How?** Asks the Tutor to give explicit instruction on how to carry out the suggested next action.
 - **Why?** Asks the Tutor to give a rationale for why the suggested next action is reasonable or appropriate.

[Previous](#)

[Index](#)

[Next](#)

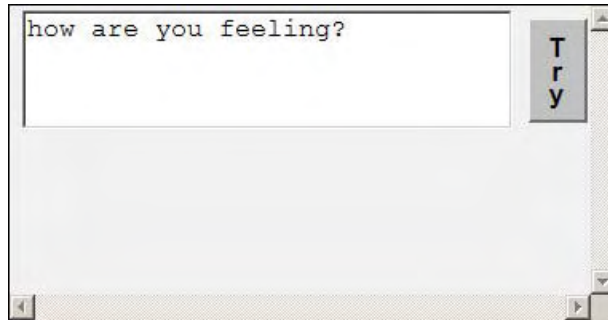
13.7 *Conversing with a Character*

[Previous](#)

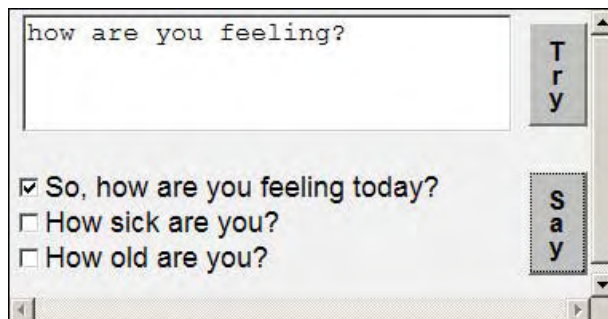
[Index](#)

[Next](#)

The largest parts of the Character and Tutor Input areas are devoted to text-based input. The area usually contains a type-in box followed by a **"Try"** button. Type text you want to say to the currently selected character, and then click **"Try"**.



In response, the system will list its best interpretations of what you said. You can choose which interpretation(s) you meant using the provided check-boxes. Once you have made your choice(s), click the **"Say"** button.



If none of the interpretations reflect what you meant to say (or if the system is unable to produce any interpretations at all), then you can edit the text and click **"Try"** again.

- **To Say Something:** Type whatever text you want then click the **"Try"** button; adjust the resulting check-box or radio-button selections and then click **"Say"**.
- **To Reject all Interpretations:** Simply go back to editing your text, and click **"Try"** again to see if the system does a better job of interpreting your modified input.

[Previous](#)

[Index](#)

[Next](#)

13.8 Accessing Patient Charts

[Previous](#)

[Index](#)

[Next](#)

To view a continuously updated patient chart click on the **"Chart"** button (the first of six buttons in the *Character Input* area when the selected character is a patient).

The **"Patient Chart"** window displays data that accumulates automatically in response to events in the simulation. This includes actions you may take, such as *asking the patient interview questions*, *examining some part of the patient*, or *ordering tests*. Sometimes it will be actions taken by other simulated agents, such as the admitting nurse filling out the cover sheet, or other staff keeping a regular record of a patient's vital signs.

The chart has seven tabs across the top. The screen below shows Chart with the Admissions tab selected:

Patient Chart	
Admission Complaint History Exam Vitals Tests Treatments	
Account	123456789
Name	Ryan Smith
Street Address	123 Main Street
City	Boston, MA
Phone	617-555-1234
SSN	123-45-6789
DOB	9/18/1960
Age	48
Sex	Male
Marital Status	Married
Religion	Buddhism
Reason for Visit	Severe fever
Accompanied By	Wife
Acc. Date/Time	2/12/2008 18:47

1. **Admission:** As shown above, contains basic information about the patient, as might appear on the admission form prepared by a triage nurse.
2. **Complaint:** Accumulates notes on the current complaint based on answers the patient gives to diagnostic interview questions.
3. **History:** Accumulates notes on the patient's medical history based on answers the patient gives to diagnostic interview questions.
4. **Exam:** Accumulates notes on the patient's physical condition based on information uncovered by performing a simulated physical examination (see the **"Examine"** button write-up).
5. **Vitals:** Accumulates a table of vital sign readings based on automatically performed measurements.
6. **Tests:** Accumulates a record of ordered tests and their results. Note that test results generally do not arrive immediately upon being ordered, but are delayed until they might realistically be ready.
7. **Treatments:** Accumulates a record of ordered treatments.

[Previous](#)

[Index](#)

[Next](#)

13.9 Examining a Patient

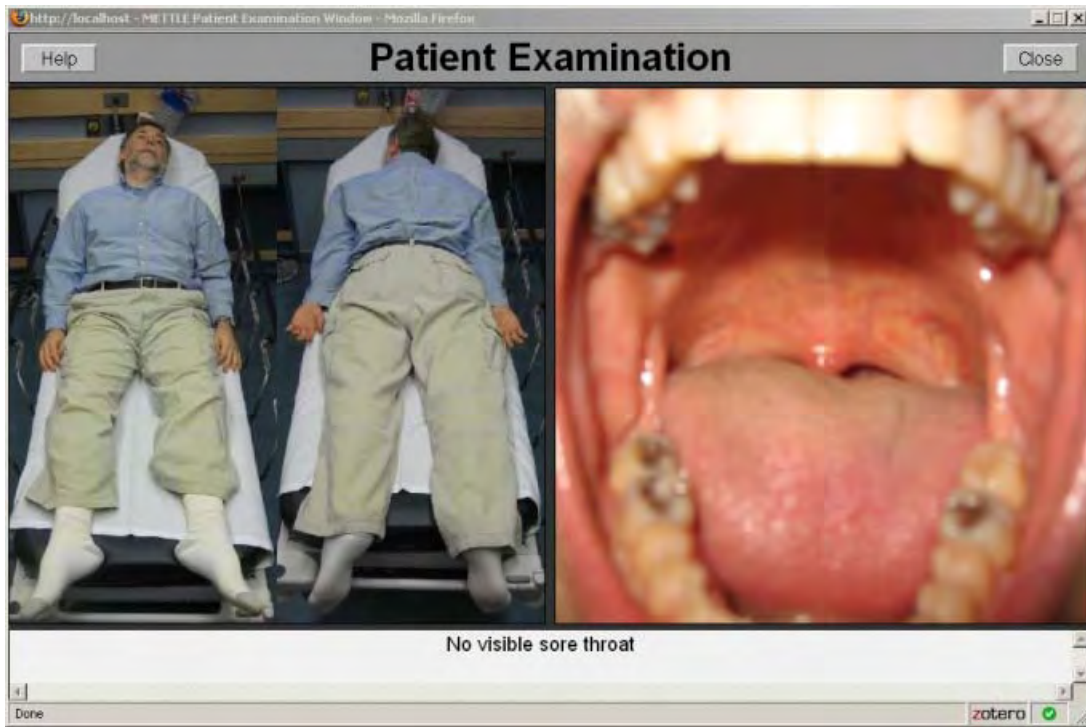
[Previous](#)

[Index](#)

[Next](#)

To conduct a simulated physical examination click on the **"Examine"** button (the second of six buttons in the *Character Input* area when the selected character is a patient).

The **"Patient Examination"** window is comprised of three main areas: (1) the left half of the window contains a side-by-side front-and-back image of the entire patient containing many mouse-sensitive regions that can be clicked on, (2) the right half of the window contains an area reserved for detailed images produced by clicking on those patient body parts, and (3) the bottom of the window displays important information that would be revealed by examining the selected part of the patient's body.



- **To Examine a Patient Body Part:** As you run the mouse over the side-by-side patient overview images, a series of hot-spots will light up in yellow. Clicking on such hot-spots will generally produce a detailed image in the area to the right. In the event that there is a relevant finding produced by examining that part of the patient, a summary will be displayed at the bottom of the window.

[Previous](#)

[Index](#)

[Next](#)

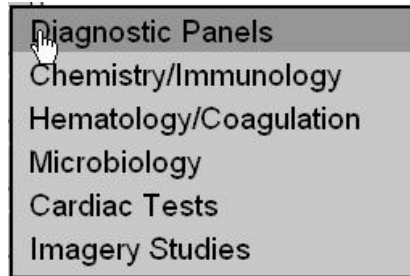
13.10 Ordering Patient Tests

[Previous](#)

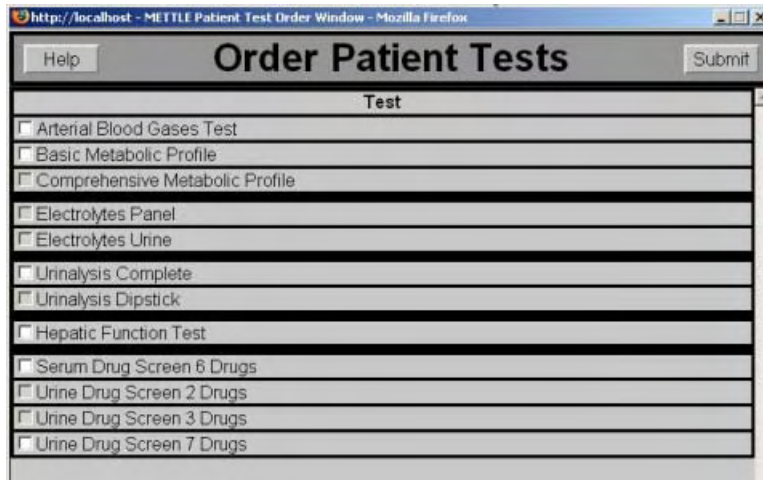
[Index](#)

[Next](#)

To order medical tests for the simulated patient click on the **"Test"** button (the third of six buttons in the *Character Input* area when the selected character is a patient). This will produce a pop-up menu offering several general categories of tests (e.g. blood tests, imagery studies, etc.).



The **"Order Patient Tests"** window will contain a form appropriate to ordering the class of tests chosen. Generally, it will contain a collection of labeled check-boxes; simply check the boxes for the tests you want to order. Once you have made your selections, click the **"Submit"** button in the upper right corner of the window.

A screenshot of a web browser window showing a form titled 'Order Patient Tests'. The browser's address bar shows 'http://localhost - METTLE Patient Test Order Window - Mozilla Firefox'. The form has a header bar with 'Help' on the left and 'Submit' on the right. Below the header is a table with a single column titled 'Test'. The table contains ten rows, each with a checkbox and a test name: 'Arterial Blood Gases Test', 'Basic Metabolic Profile', 'Comprehensive Metabolic Profile', 'Electrolytes Panel', 'Electrolytes Urine', 'Urinalysis Complete', 'Urinalysis Dipstick', 'Hepatic Function Test', 'Serum Drug Screen 6 Drugs', and 'Urine Drug Screen 2 Drugs'. The last two rows are partially visible.

[Previous](#)

[Index](#)

[Next](#)

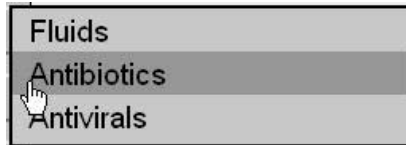
13.11 Ordering Patient Treatments

[Previous](#)

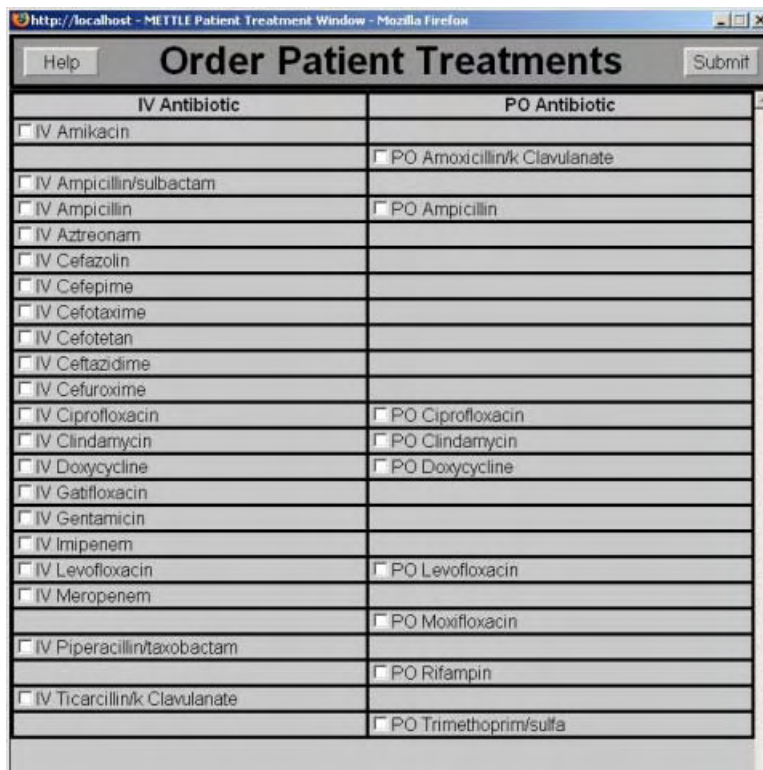
[Index](#)

[Next](#)

To order medical treatments for the simulated patient click on the **"Treat"** button (the fourth of six buttons in the *Character Input* area when the selected character is a patient). This will produce a pop-up menu offering several general categories of treatments (e.g. administration of fluids, antibiotics, and antivirals).



The **"Order Patient Treatments"** window will contain a form appropriate to ordering the class of treatments chosen. Generally, it will contain a collection of labeled check-boxes; simply check the boxes for the treatments you want to order. Once you have made your selections, click the **"Submit"** button in the upper right corner of the window.



IV Antibiotic	PO Antibiotic
<input type="checkbox"/> IV Amikacin	<input type="checkbox"/> PO Amoxicillin/k Clavulanate
<input type="checkbox"/> IV Ampicillin/sulbactam	<input type="checkbox"/> PO Ampicillin
<input type="checkbox"/> IV Ampicillin	
<input type="checkbox"/> IV Aztreonam	
<input type="checkbox"/> IV Cefazolin	
<input type="checkbox"/> IV Cefepime	
<input type="checkbox"/> IV Cefotaxime	
<input type="checkbox"/> IV Cefotetan	
<input type="checkbox"/> IV Ceftazidime	
<input type="checkbox"/> IV Cefuroxime	
<input type="checkbox"/> IV Ciprofloxacin	<input type="checkbox"/> PO Ciprofloxacin
<input type="checkbox"/> IV Clindamycin	<input type="checkbox"/> PO Clindamycin
<input type="checkbox"/> IV Doxycycline	<input type="checkbox"/> PO Doxycycline
<input type="checkbox"/> IV Gabfloxacin	
<input type="checkbox"/> IV Gentamicin	
<input type="checkbox"/> IV Imipenem	
<input type="checkbox"/> IV Levofloxacin	<input type="checkbox"/> PO Levofloxacin
<input type="checkbox"/> IV Meropenem	
	<input type="checkbox"/> PO Moxifloxacin
<input type="checkbox"/> IV Piperacillin/tazobactam	<input type="checkbox"/> PO Rifampin
<input type="checkbox"/> IV Ticarcillin/k Clavulanate	<input type="checkbox"/> PO Trimethoprim/sulfa

[Previous](#)

[Index](#)

[Next](#)

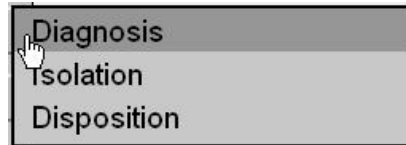
13.12 Patient Management Decisions

[Previous](#)

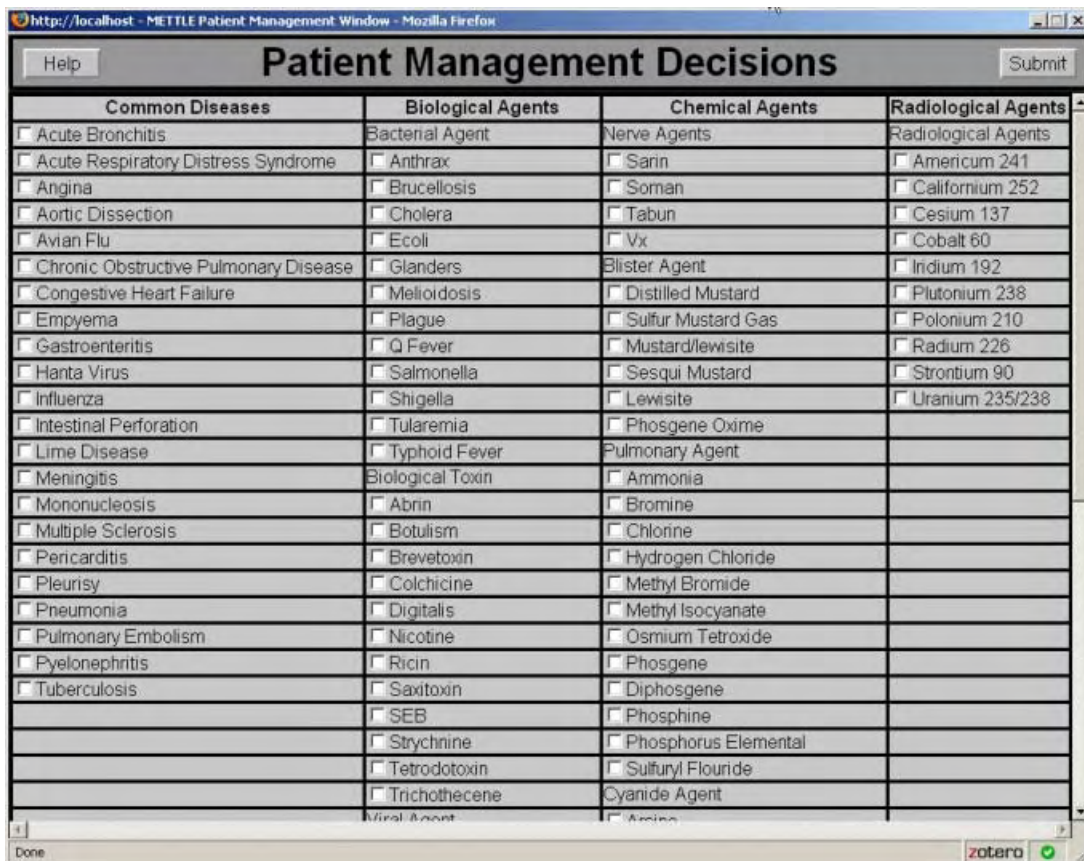
[Index](#)

[Next](#)

To indicate decisions about aspects of patient management other than tests and treatments click on the **"Manage"** button (the fifth of six buttons in the *Character Input* area when the selected character is a patient). This will produce a pop-up menu offering several general categories of management decisions (e.g. diagnosis, isolation, and disposition).



The **"Patient Management Decisions"** window will contain a form appropriate to registering the class of decision chosen. Generally, it will contain a collection of labeled check-boxes; simply check the boxes for the decisions you want to order. Once you have made your selections, click the **"Submit"** button in the upper right corner of the window.



Common Diseases	Biological Agents	Chemical Agents	Radiological Agents
<input type="checkbox"/> Acute Bronchitis	Bacterial Agent	Nerve Agents	Radiological Agents
<input type="checkbox"/> Acute Respiratory Distress Syndrome	<input type="checkbox"/> Anthrax	<input type="checkbox"/> Sarin	<input type="checkbox"/> Americium 241
<input type="checkbox"/> Angina	<input type="checkbox"/> Brucellosis	<input type="checkbox"/> Soman	<input type="checkbox"/> Californium 252
<input type="checkbox"/> Aortic Dissection	<input type="checkbox"/> Cholera	<input type="checkbox"/> Tabun	<input type="checkbox"/> Cesium 137
<input type="checkbox"/> Avian Flu	<input type="checkbox"/> Ecoli	<input type="checkbox"/> Vx	<input type="checkbox"/> Cobalt 60
<input type="checkbox"/> Chronic Obstructive Pulmonary Disease	<input type="checkbox"/> Glanders	Blister Agent	<input type="checkbox"/> Iridium 192
<input type="checkbox"/> Congestive Heart Failure	<input type="checkbox"/> Melioidosis	<input type="checkbox"/> Distilled Mustard	<input type="checkbox"/> Plutonium 238
<input type="checkbox"/> Empyema	<input type="checkbox"/> Plague	<input type="checkbox"/> Sulfur Mustard Gas	<input type="checkbox"/> Polonium 210
<input type="checkbox"/> Gastroenteritis	<input type="checkbox"/> Q Fever	<input type="checkbox"/> Mustard/Lewisite	<input type="checkbox"/> Radium 226
<input type="checkbox"/> Hanta Virus	<input type="checkbox"/> Salmonella	<input type="checkbox"/> Sesqui Mustard	<input type="checkbox"/> Strontium 90
<input type="checkbox"/> Influenza	<input type="checkbox"/> Shigella	<input type="checkbox"/> Lewisite	<input type="checkbox"/> Uranium 235/238
<input type="checkbox"/> Intestinal Perforation	<input type="checkbox"/> Tularemia	<input type="checkbox"/> Phosgene Oxime	
<input type="checkbox"/> Lyme Disease	<input type="checkbox"/> Typhoid Fever	Pulmonary Agent	
<input type="checkbox"/> Meningitis	Biological Toxin	<input type="checkbox"/> Ammonia	
<input type="checkbox"/> Mononucleosis	<input type="checkbox"/> Abrin	<input type="checkbox"/> Bromine	
<input type="checkbox"/> Multiple Sclerosis	<input type="checkbox"/> Botulism	<input type="checkbox"/> Chlorine	
<input type="checkbox"/> Pericarditis	<input type="checkbox"/> Brevetoxin	<input type="checkbox"/> Hydrogen Chloride	
<input type="checkbox"/> Pleurisy	<input type="checkbox"/> Colchicine	<input type="checkbox"/> Methyl Bromide	
<input type="checkbox"/> Pneumonia	<input type="checkbox"/> Digitalis	<input type="checkbox"/> Methyl Isocyanate	
<input type="checkbox"/> Pulmonary Embolism	<input type="checkbox"/> Nicotine	<input type="checkbox"/> Osmium Tetroxide	
<input type="checkbox"/> Pyelonephritis	<input type="checkbox"/> Ricin	<input type="checkbox"/> Phosgene	
<input type="checkbox"/> Tuberculosis	<input type="checkbox"/> Saxitoxin	<input type="checkbox"/> Diphosgene	
	<input type="checkbox"/> SEB	<input type="checkbox"/> Phosphine	
	<input type="checkbox"/> Strychnine	<input type="checkbox"/> Phosphorus Elemental	
	<input type="checkbox"/> Tetrodotoxin	<input type="checkbox"/> Sulfuryl Fluoride	
	<input type="checkbox"/> Trichothecene	Cyanide Agent	
	Viral Agent	<input type="checkbox"/> Arsenic	

[Previous](#)

[Index](#)

[Next](#)

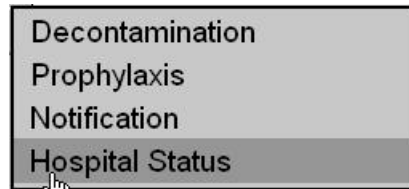
13.13 Emergency Systemic Response

[Previous](#)

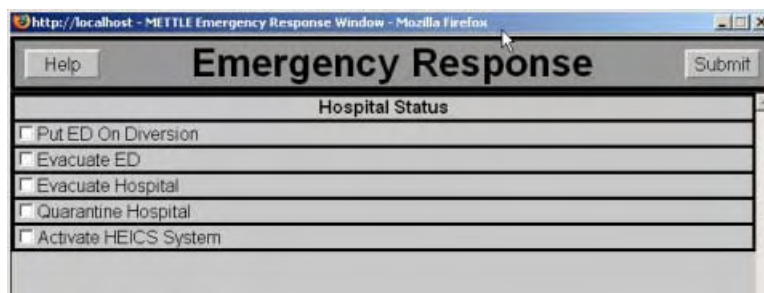
[Index](#)

[Next](#)

To indicate decisions about how the hospital should respond to an evolving emergency situation click on the "**Respond**" button (the last of six buttons in the *Character Input* area when the selected character is a patient). This will produce a pop-up menu offering several general categories of response decisions (e.g. decontamination, prophylaxis, and notifications).



The "**Emergency Response**" window will contain a form appropriate to registering the class of decision chosen. Generally, it will contain a collection of labeled check-boxes; simply check the boxes for the decisions you want to order. Once you have made your selections, click the "**Submit**" button in the upper right corner of the window.



[Previous](#)

[Index](#)

[Next](#)

13.14 Reviewing Scenario Play

[Previous](#)

[Index](#)

[Next](#)

At the end of scenario play, you will be shown the "Scorecard" window which allows you to see how you have been doing with respect to the system's evaluations of performance on its curriculum. The four sections of the screen layout are shown and discussed below.

The screenshot shows a web browser window titled "http://localhost - METTLE Score Card - Mozilla Firefox". The main content area is titled "METTLE Score Card" and contains four distinct sections outlined by a red border:

- 1. Scenario History:** A table with columns "Exercise" and "When Run". It lists "Overall" and "CBR-1-Anthrax" with a timestamp "03/19/2008-13:59:24".
- 2. Selected Scenario Summary:** A summary for the selected scenario "CBR 1 Anthrax", describing a patient in an emergency department with an unknown illness.
- 3. Curriculum Hierarchy w. Scores:** A table with columns "Wins", "Losses", and "Curriculum Point". It shows scores for "ED Practice", "ED Sequencing", and "ED Sequencing IVsEarly".
- 4. Selected Curriculum Item:** Details for the selected curriculum point "ED Sequencing IVsEarly", including a description "Get IVs placed early".

1. **Scenario History:** Contains a table listing all of the scenarios you have played and when you played them. When you click on an item in this table, the summary of that scenario appears below, and your performance on relevant curriculum points is displayed to the right. Note that the first item in the table provides a way to see your curriculum scores aggregated over *all* scenarios you have played.
2. **Selected Scenario Summary:** Displays the summary information for the scenario currently selected in the Scenario History above. This is just intended to help you remember which scenarios are which, as you build up a longer history with the system.
3. **Curriculum Hierarchy with Scores:** Contains a tree showing those parts of the curriculum hierarchy that were touched on when you ran the selected scenario. For each item, it shows "Wins" and "Losses," that is, counts of how often you demonstrated mastery of the curriculum element, versus how often you failed to apply it or had to ask for help; in each column, the number before the slash represents the counts from this scenario, and the number after the slash represents the counts across all scenarios you have played. When you click on an item in the tree, details of the selected curriculum point are displayed below.

4. **Selected Curriculum Item:** Contains information about the curriculum item selected in the tree above. In a more fully fleshed out system, this summary would contain hyperlinks for accessing instructional materials related to the selected curriculum point.

[Previous](#)

[Index](#)

[Next](#)

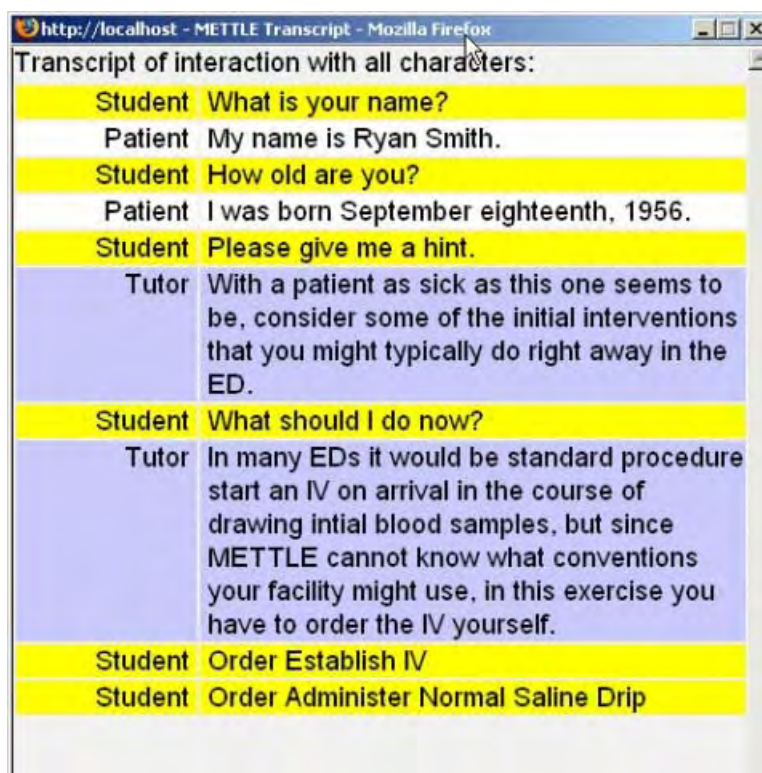
13.15 Transcript Windows

[Previous](#)

[Index](#)

[Next](#)

At any point during scenario play you can request to see a history of your interactions so far. If you click the "**History**" button at the top of the Scenario Player screen, you will see a complete history of what has happened so far. Alternately, you can click on the images of the currently selected simulated actor on the left, or the simulated Tutor on the right, to see just the history of your interaction with that character.



Transcript of interaction with all characters:	
Student	What is your name?
Patient	My name is Ryan Smith.
Student	How old are you?
Patient	I was born September eighteenth, 1956.
Student	Please give me a hint.
Tutor	With a patient as sick as this one seems to be, consider some of the initial interventions that you might typically do right away in the ED.
Student	What should I do now?
Tutor	In many EDs it would be standard procedure start an IV on arrival in the course of drawing initial blood samples, but since METTLE cannot know what conventions your facility might use, in this exercise you have to order the IV yourself.
Student	Order Establish IV
Student	Order Administer Normal Saline Drip

Items in the history appear in the order they happened. To see the earliest actions you may have to scroll back to the top of the screen. The backgrounds of the transcript items are color-coded:

- **Yellow:** Your actions as student.
- **White:** Actions of some simulated character.
- **Blue:** Action by the simulated Tutor.

Note that clicks on the four "**Hint**", "**What**", "**How**", and "**Why**" Tutor buttons are translated into more complete sentences you might have uttered to ask those questions.

[Previous](#)

[Index](#)

[Next](#)

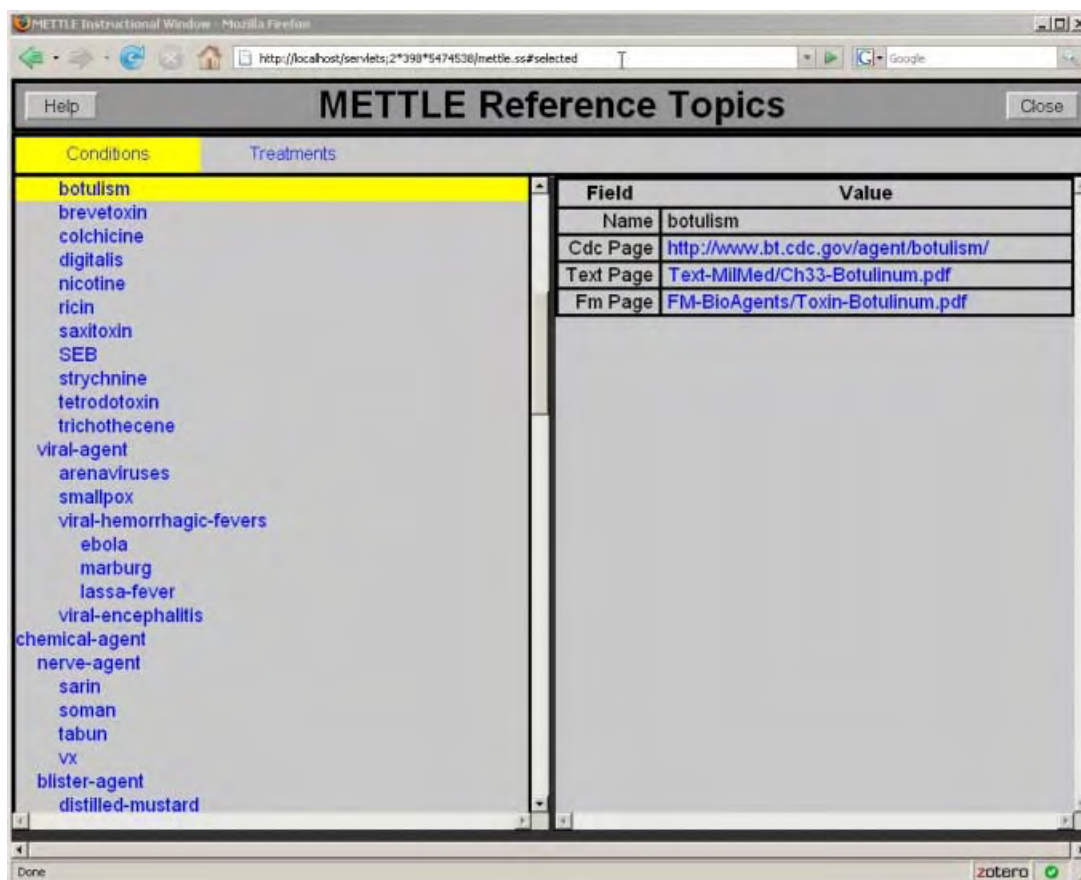
13.16 Reference Window

[Previous](#)

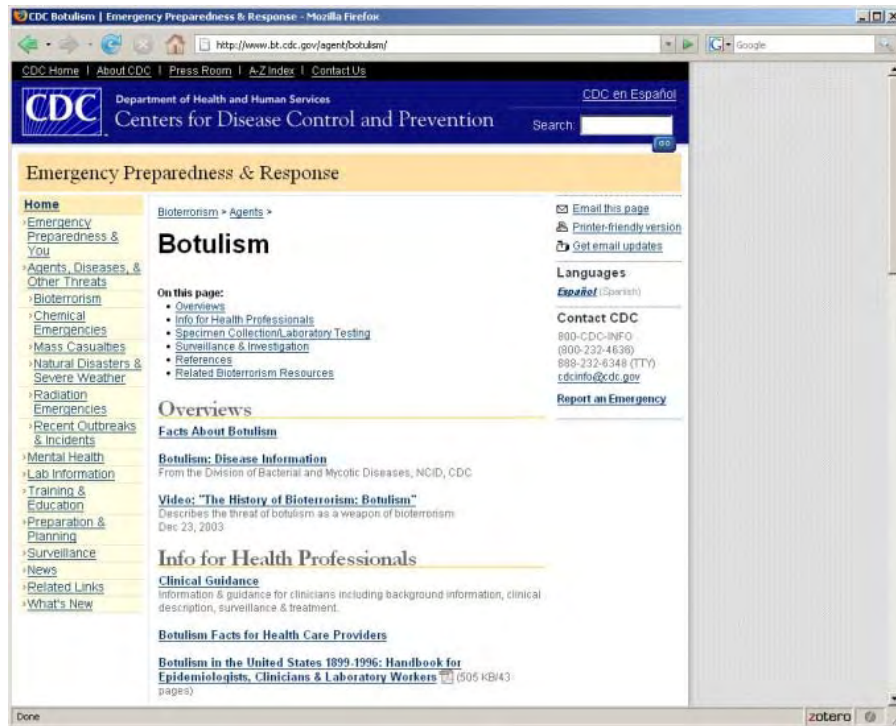
[Index](#)

[Next](#)

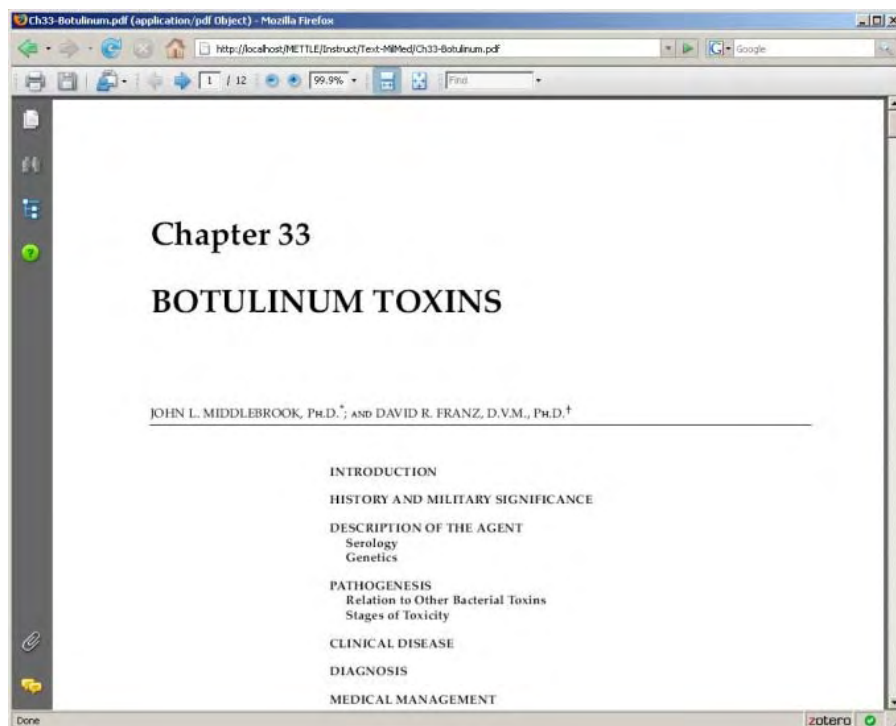
At any point during scenario play you can request to see reference materials that have been linked into the system. If you click the **"Reference"** button at the top of the Scenario Player screen, you will see a window that offers tabs such as **"Conditions"** and **"Treatments"**. Each such tab displays two panes: a tree of relevant concepts appears on the left, while details for the selected concept from that tree appear on the right. Those details will generally be a list of links to authoritative sources of information on the concept.



Some links may take you to live pages accessible on the internet, as at the CDC:



Other links may take you to information stored as part of the METTLE system, including excerpts from (military) medical texts.



[Previous](#)

[Index](#)

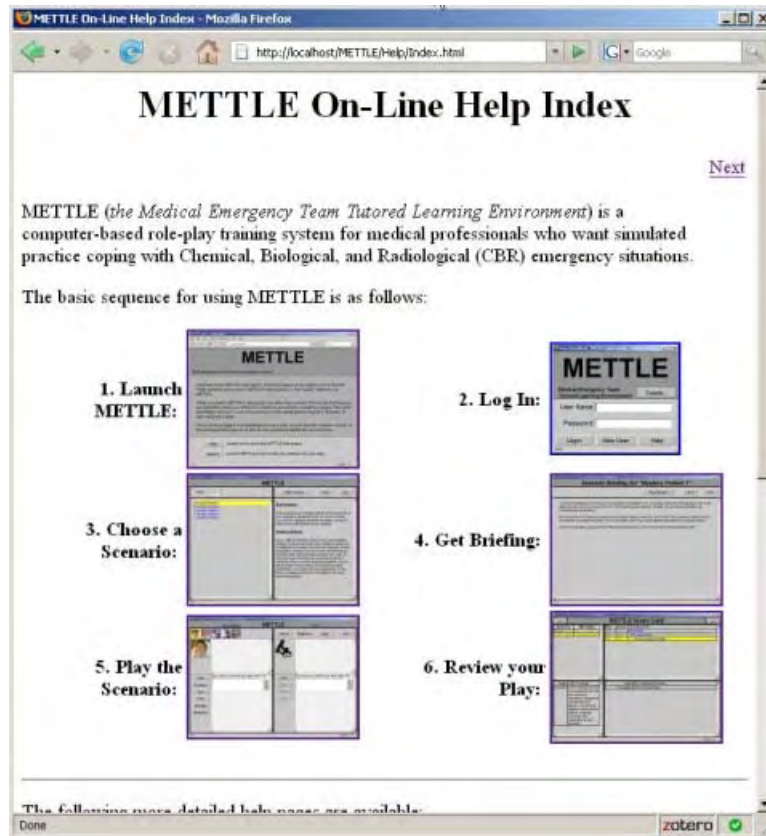
[Next](#)

13.17 Help Window

[Previous](#)

[Index](#)

Most windows in the METTLE system provide a "**Help**" button that will bring you to a relevant part of the network of Help pages. Help screens generally have a set of links at the top and bottom of the page that allow you to move the a "**Previous**" or "**Next**" screen, and an "**Index**" link that takes you to the Help system index page. The Help Index page is a good place to look if the page you are on is not providing the help you need.



[Previous](#)

[Index](#)

14 Appendix H: Final Evaluation Feedback Form

METTLE Evaluation Study Feedback Form

Purpose The prototype Medical Emergency Team Tutored Learning Environment (METTLE) enables students to play the role of an emergency room physician confronted with a patient with an unknown illness. The system simulates interactions with the patient and other medical personnel, and provides automated performance feedback to the student.

We are very interested in your opinions about this prototype system. Information you provide on this form will help us identify the strengths and weaknesses of the prototype to guide the development of future versions of this and other systems. Your feedback will also help us assess the overall effectiveness of the instructional methods and technologies used. This information may be reported to our research sponsor, the Department of Defense, through the managing agency, the USAMRMC Telemedicine and Advanced Technology Research Center (TATRC).

Confidentiality Your identity will not be disclosed to any persons or organizations outside of the research project team, except for authorized representatives of the USAMRMC.

Contacting Us This prototype was developed by a research project team led by Stottler Henke Associates, Inc, headquartered in San Mateo, CA, with most of the project work performed at the company office in Somerville, MA. For additional information, please contact the research project manager, Dr. Eric Domeshek, at 617-616-1291.

Email: domeshek@stottlerhenke.com. Web: <http://www.stottlerhenke.com>.

1. Your expertise

Please characterize your **specialty** (circle any that apply, and/or fill in "Other"):

Internal Medicine Emergency Medicine Infectious Diseases Toxicology
Radiology Pathology Other: _____

Please characterize your **seniority** (number of years post-bachelors in your profession): _____

Please characterize your areas of **relevant experience** (mark a position on each scale below):

Emergency medicine	1	2	3	4	5
Developing medical education/training	1	2	3	4	5
Delivering medical education/training	1	2	3	4	5
Computer-Based Training (CBT) use	1	2	3	4	5
CBT development	1	2	3	4	5
	Not experienced				Very experienced

(Over)

2. Relative effectiveness of the instructional method

Please enter your perception of the likely effectiveness of *instructional approaches* illustrated by this prototype training simulation for the following learning objectives, when compared to conventional approaches such as attending lectures or reading articles:

Become exposed to uncommon emergency situations and relevant knowledge & skills	1	2	3	4	5
Practice applying emergency medical response knowledge & skills	1	2	3	4	5
Acquire skill proficiency at emergency medical response	1	2	3	4	5
Identify emergency medical response knowledge & skill gaps	1	2	3	4	5
	Much less effective		As effective	Much more effective	

3. Effectiveness of specific aspects of the training system

Please enter your perception of the effectiveness of specific aspects of the prototype:

Scenario situations and events challenge students to apply medical response concepts and skills	1	2	3	4	5
Simulated patient interaction provides decision cues that prompt students to assess the situation & make decisions	1	2	3	4	5
The simulated patient chart organizes data to support students in situation assessment & decision-making	1	2	3	4	5
Other simulated conversations provide useful cues that support students in decision-making	1	2	3	4	5
Simulated interactions let students select actions demonstrating medical response knowledge and skills	1	2	3	4	5
Simulated tutor conversations let student express and explore their rationale for decision-making	1	2	3	4	5
Automated performance feedback helps students learn and identify knowledge and skill gaps	1	2	3	4	5
	Not effective		Somewhat effective	Highly effective	

4. Time Spent on METTLE Evaluation

Please estimate the amount of time you spent on each phase of the METTLE evaluation:

Setting up the software

Reviewing video and/or help pages

Using the simulation

Filling out this questionnaire

5. Other comments

Please use this sheet to describe any other unique or significant strengths or weaknesses of the system.
Or, provide any other comments you may have.